Remove Duplicate Letters

Given a string which contains only lowercase letters, remove duplicate letters so that every letter appear once and only once. You must make sure your result is the smallest in lexicographical order among all possible results.

Example:

Given `"bcabc"`
Return `"abc"`

Given `"cbacdcbc"`
Return `"acdb"`

Credits:
Special thanks to @dietpepsi for adding this problem and creating all test cases.

Solution 1

Given the string s, the greedy choice (i.e., the leftmost letter in the answer) is the smallest s[i], s.t. the suffix s[i ..] contains all the unique letters. (Note that, when there are more than one smallest s[i]'s, we choose the leftmost one. Why? Simply consider the example: "abcacb".)

After determining the greedy choice s[i], we get a new string s' from s by

1. removing all letters to the left of s[i],
2. removing all s[i]'s from s.

We then recursively solve the problem w.r.t. s'.

The runtime is O(26 * n) = O(n).

```
public class Solution {
    public String removeDuplicateLetters(String s) {
        int[] cnt = new int[26];
        int pos = 0; // the position for the smallest s[i]
        for (int i = 0; i < s.length(); i++) cnt[s.charAt(i) - 'a']++;
        for (int i = 0; i < s.length(); i++) {
            if (s.charAt(i) < s.charAt(pos)) pos = i;
            if (--cnt[s.charAt(i) - 'a'] == 0) break;
        }
        return s.length() == 0 ? "" : s.charAt(pos) + removeDuplicateLetters(s.su
bstring(pos + 1).replaceAll("" + s.charAt(pos), ""));
    }
}
```

written by lixx2100 original link here

Solution 2

The basic idea is to find out the smallest result letter by letter (one letter at a time). Here is the thinking process for input "cbacdcbc":

1. find out the last appeared position for each letter; c - 7 b - 6 a - 2 d - 4
2. find out the smallest index from the map in step 1 (a - 2);
3. the first letter in the final result must be the smallest letter from index 0 to index 2;
4. repeat step 2 to 3 to find out remaining letters.

- the smallest letter from index 0 to index 2: a
- the smallest letter from index 3 to index 4: c
- the smallest letter from index 4 to index 4: d
- the smallest letter from index 5 to index 6: b

so the result is "acdb"

Notes:

- after one letter is determined in step 3, it need to be removed from the "last appeared position map", and the same letter should be ignored in the following steps
- in step 3, the beginning index of the search range should be the index of previous determined letter plus one

---

```java
public class Solution {

    public String removeDuplicateLetters(String s) {
        if (s == null || s.length() <= 1) return s;

        Map<Character, Integer> lastPosMap = new HashMap<>();
        for (int i = 0; i < s.length(); i++) {
            lastPosMap.put(s.charAt(i), i);
        }

        char[] result = new char[lastPosMap.size()];
        int begin = 0, end = findMinLastPos(lastPosMap);

        for (int i = 0; i < result.length; i++) {
            char minChar = 'z' + 1;
            for (int k = begin; k <= end; k++) {
                if (lastPosMap.containsKey(s.charAt(k)) && s.charAt(k) < minChar)
{
                    minChar = s.charAt(k);
                    begin = k+1;
                }
            }

            result[i] = minChar;
            if (i == result.length-1) break;

            lastPosMap.remove(minChar);
            if (s.charAt(end) == minChar) end = findMinLastPos(lastPosMap);
        }

        return new String(result);
    }

    private int findMinLastPos(Map<Character, Integer> lastPosMap) {
        if (lastPosMap == null || lastPosMap.isEmpty()) return -1;
        int minLastPos = Integer.MAX_VALUE;
        for (int lastPos : lastPosMap.values()) {
             minLastPos = Math.min(minLastPos, lastPos);
        }
        return minLastPos;
    }

}
```

written by WHJ425 original link here

Solution 3

for "cbacdcbc", we counts each letter's index:

```
a----2
b----1,6
c----0,3,5,7
d----4
```

we go from a to d, to find the first letter who has a index smaller than the largest index of the rest. Here, index 2 of letter a is smaller than 6, 7, 4, so we first pick a; then we remove all index smaller than 2, and we have:

```
b----6
c----3,5,7
d----4
```

the next round we pick c not b, why ? cuz 6 of b is larger than 4, but 3 of c is smaller than 4 and 6.

```
b---6
d---4
```

then we pick d and b to form "acdb"

O(n) time to count index, and as we only have 26 letters, it's about O(26 * 26) to find a candidate letter and O(n) time to remove all index. So I think the running time is O(n).

```java
public class Solution {
    public String removeDuplicateLetters(String s) {
        HashMap<Character, ArrayList<Integer>> counts = new HashMap<Character, Ar
rayList<Integer>>();
        ArrayList<Character> keys = new ArrayList<Character>();
        for (int i = 0; i < s.length(); i++) {
            char c = s.charAt(i);
            if (!counts.containsKey(c)) {
                counts.put(c, new ArrayList<Integer>());
                keys.add(c);
            }
            counts.get(c).add(i);
        }
        Collections.sort(keys);
        StringBuilder sb = new StringBuilder();
        while (!counts.isEmpty()) {
            boolean found = true;
            for (int i = 0; i < keys.size(); i++) {
                int index = counts.get(keys.get(i)).get(0);
                for (int j = 0; j < keys.size(); j++) {
                    ArrayList<Integer> count = counts.get(keys.get(j));
                    if (count.get(count.size() - 1) < index) {
                        found = false;
                        break;
                    }
                }
                if (found) {
                    sb.append(keys.get(i));
                    counts.remove(keys.get(i));
                    keys.remove(i);
                    for (int j = 0; j < keys.size(); j++) {
                        ArrayList<Integer> count = counts.get(keys.get(j));
                        while (count.get(0) < index) {
                            count.remove(0);
                        }
                    }
                    break;
                }
                found = true;
            }
        }
        return sb.toString();
    }
}
```

written by mayanist original link here