拉勾教育
—互联网人实战大学—

栈+贪心算法：去除重复字母

困难/栈、贪心算法

学习目标

拉勾教育
—互联网人实战大学—

- 加深对贪心算法原理的理解
- 巩固在实际问题中对贪心算法的应用
- 结合数据结构栈完成对字符串的处理

题目描述

给你一个仅包含小写字母的字符串，请你去除字符串中重复的字母，使得每个字母只出现一次。需保证返回结果的字典序最小（要求不能打乱其他字符的相对位置）。

输入: "bcabc"
输出: "abc"

输入: "cbacdcbc"
输出: "acdb"

一. Comprehend 理解题意

拉勾教育
—互联网人实战大学—

题目主干

去除字符串中重复的字母，使得每个字母只出现一次

- 如果某个字母在字符串中出现多次，我们需要删去重复的字母

"bcabc"

- 由于右侧字符串中只有a,b,c三个字母，那么去除重复字母后的字符串可能为“abc”，“bca”，“cab”，“bac”，“cba”，“acb”

附加限制

1. 不能打乱字符的相对位置：“cba”，“acb”不满足条件（无法通过删除字符得到）
2. 返回的字典序最小：以a开头的字符串字典序一定小于以其他字母开头的字符串

有 “abc” < “bac” < “bca” < “cab” ，故应返回“abc”

一. Comprehend 理解题意

宽松限制

输入仅包含小写字母a-z的字符串

细节问题

"dcbadd"

最终返回的字符串可能不以字符串中出现的最小字母开头

如：最小字符为a，但是符合题意的字符串应为 "cbad"

二. Choose 数据结构及算法思维选择

数据结构选择

- 输入的数据类型为一个字符串
- 输出的数据类型为去除重复字母的字符串
- 因为需要对字符串进行处理，我们使用字符串作为基本数据结构

二. Choose 数据结构及算法思维选择

算法思维选择

- 题目要求寻找字典序最小的字符串

"dcada"

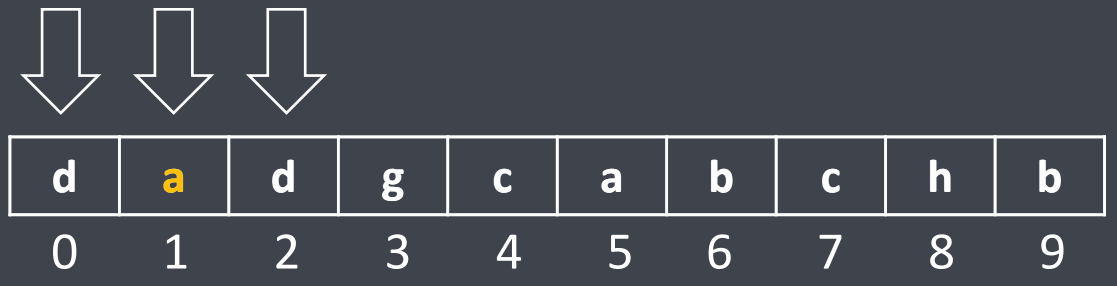
  - 每次都选择尽可能的小字符作为本轮的首字符
  - 同时要保证本轮选择的首字符前的其他字符在首字符后出现过
- 示例字符串中最小的字符为a
  - 将a放在返回字符串的第一个，字典序一定最小
  - 但字符c并未在原字符串任意一个字符a之后出现，不满足保持相对关系的限制
  - 满足上面两个条件的字符应为c，所以返回的字符串第一个字符是c

三. Code 基本解法及编码实现

拉勾教育
—互联网人实战大学—

解题思路剖析

- 先浏览一遍字符串，确定每个字符出现的最后位置
- 当前最小字符为 d
- 当前字符是否是最后一次出现：是
- 最小字符为a
- 取出a后面的子串，删去重复出现的a



Last Occurrence

a	b	c	d	g	h
5	9	7	2	3	8

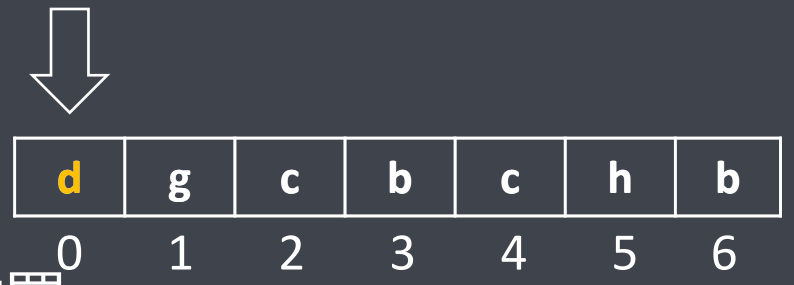
Results

a					

三. Code 基本解法及编码实现

解题思路剖析

- 继续在子串中寻找起始字符
- 遍历字符串，确定每个字符出现的最后位置
- 当前最小字符为 **d**
- 当前字符是否是最后一次出现： **是**
- 最小字符为d
- 取出d后面的子串，删去重复出现的d



Last Occurrence

b	c	d	g	h
6	4	0	1	5

Results

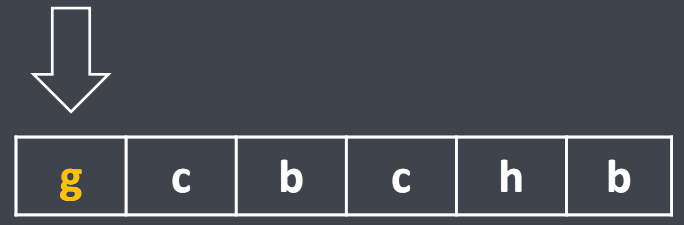
a	d				

三. Code 基本解法及编码实现

拉勾教育
—互联网人实战大学—

解题思路剖析

- 继续在子串中寻找起始字符
- 遍历字符串，确定每个字符出现的最后位置
- 当前最小字符为 **g**
- 当前字符是否是最后一次出现： **是**
- 最小字符为g
- 取出g后面的子串，删去重复出现的g



删去字符d后的子串

Last Occurrence

b	c	g	h
5	3	0	4

Results

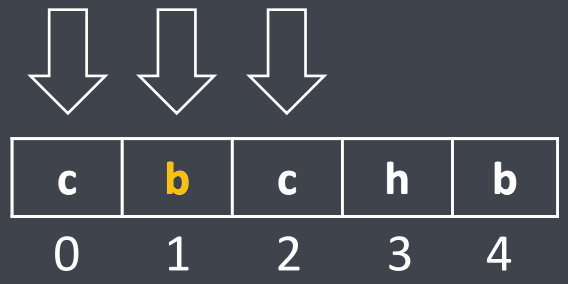
a	d	g			

三. Code 基本解法及编码实现

拉勾教育
—互联网人实战大学—

解题思路剖析

- 继续在子串中寻找起始字符
- 先浏览一遍字符串，确定每个字符出现的最后位置
- 当前最小字符为 b
- 当前字符是否是最后一次出现：是
- 最小字符为b
- 取出b后面的子串，删去重复出现的b



删去字符g后的子串

Last Occurrence

b	c	h
4	2	3

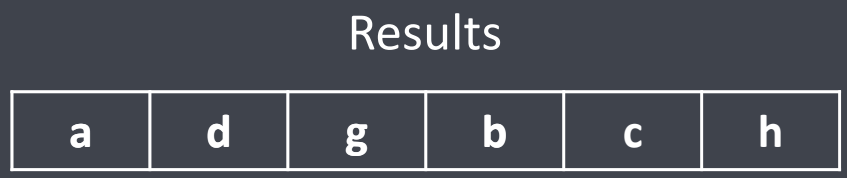
Results

a	d	g	b		

三. Code 基本解法及编码实现

解题思路剖析

- 继续在子串中寻找起始字符
- 完成任务



三. Code 基本解法及编码实现

复杂度分析

时间复杂度

- 统计每一个字符最后出现的位置，需要对字符串浏览一遍，时间复杂度为O(n)
- 递归的对字符串进行处理，每次找到起始字符，最多处理26次，每次最多浏览一遍字符串，时间复杂度为O(n)，总时间复杂度为O(n)

空间复杂度

- 空间消耗方面我们只需要存储返回的字符串（最大长度为26）和其他辅助变量，均为常数级的变量空间，空间复杂度为O(1)
- 递归调用的最大深度为26次，所以总的空间复杂度为O(1)

三. Code 基本解法及编码实现

Java编码实现

```java
public String removeDuplicateLetters(String s) {
    int length = s.length();
    //结束条件 空字符串不处理
    if (length == 0) return "";
    //记录字符串中出现过的字符在字符串最后一次出现的位置
    int[] last_occurrence = new int[26];
    //统计小写字符'a'-'z'在字符串中出现的次数
    for (int i = 0; i < length; i++)
        last_occurrence[s.charAt(i) - 'a'] = i;
    int pos = 0; //  指针记录当前处理过的字符串中最小的字符位置
    for (int i = 0; i < length; i++) {
        if (s.charAt(i) < s.charAt(pos))       pos = i;
        //遇到该字符是最后一次出现，本轮的开头最小字符就确定了，退出当前循环，
        if (last_occurrence[s.charAt(i) - 'a'] == i) break;
    }
    //剩下的pos之后的字符串中需要删除本轮次的最小字符
    String remainedString = s.substring(pos +1).replaceAll(""+s.charAt(pos),
"");
    //递归关系式：返回当前最小字符+递归处理剩下的字符串
    return s.charAt(pos) + removeDuplicateLetters(remainedString);
}
```

执行结果： 通过 显示详情 >

执行用时： 24 ms ，在所有 Java 提交中击败了 10.09% 的用户

内存消耗： 39.3 MB ，在所有 Java 提交中击败了 13.36% 的用户

四. Consider 思考更优解

是否存在无效代码或者无效空间消耗

- 递归求解需要多次遍历字符串
- 我们能不能只遍历一遍字符串就可以得到结果呢？

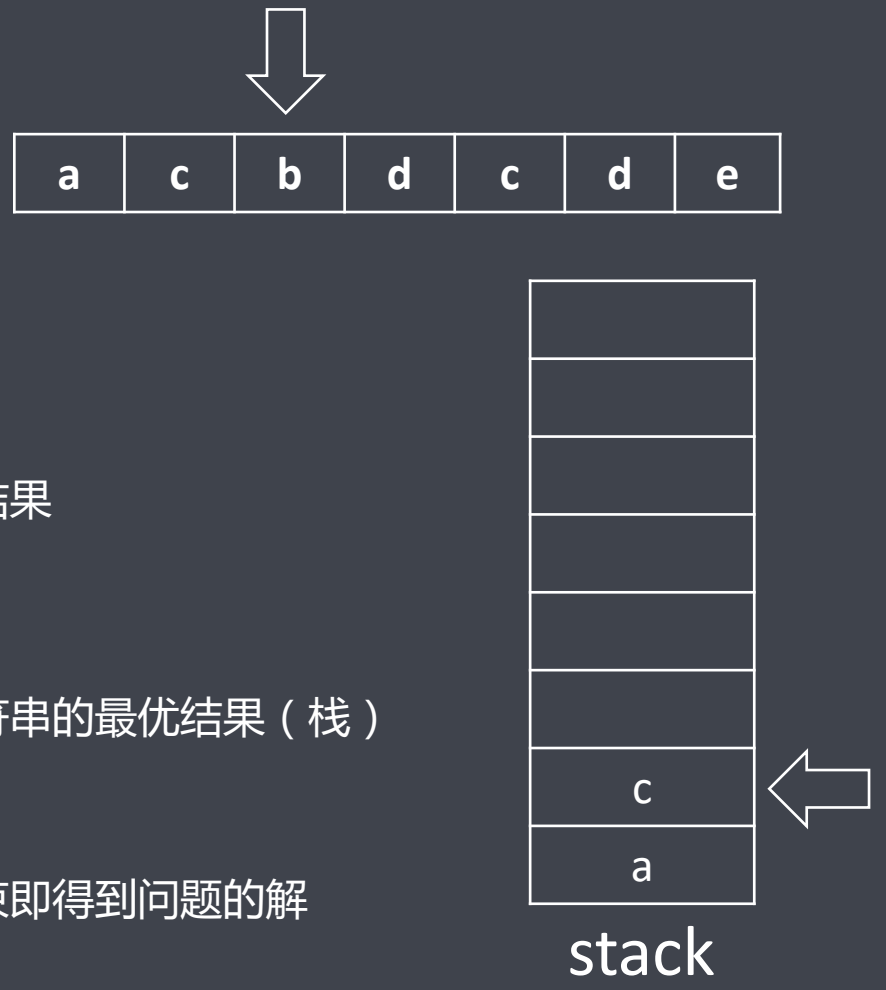
是否有更好的算法思维

- 若输入字符串为 "aaaabbbbccccddddcba"，需要遍历到 "aaaabbbbccccddddcba" 才能确定首字符为a
- 如何避免冗余的字符串扫描？

四. Consider 思考更优解

贪心算法与栈的结合

- 维护栈存放当前扫描过字符串的处理结果
- 贪心思想
  - 若当前字符小于栈顶字符且栈顶字符在之后还会出现
  - 那么使用当前字符替换栈顶字符一定可以得到字典序更小的结果
- 回顾贪心算法的流程
  - 将求解问题分成若干子问题并求解：给定当前字符及之前字符串的最优结果（栈）
  - 贪心决策当前最优解，即得到字典序更小的字符串
  - 合并子问题的解为原来问题的解：对字符串所有字符决策结束即得到问题的解



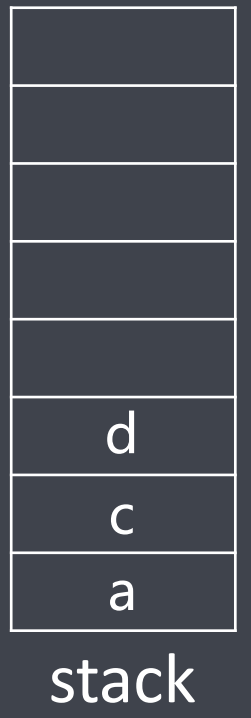
四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

贪心算法与栈的结合

示例

- 栈为空，将当前字符压栈
- 当前字符a在栈中出现过，跳过
- 当前字符c未在栈中出现，且大于栈顶字符a，压栈
- 当前字符d未在栈中出现，且大于栈顶字符c，压栈



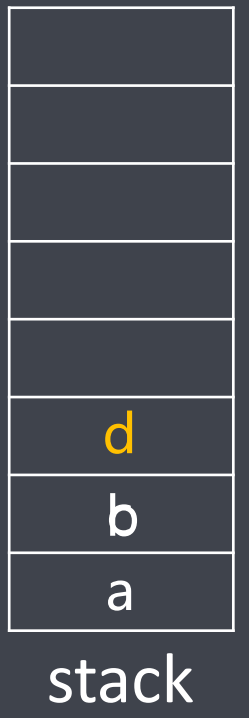
四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

贪心算法与栈的结合

示例

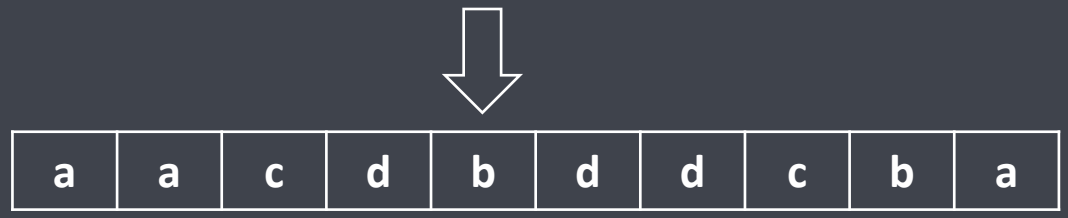
- 当前字符b小于栈顶字符d且d还会在之后出现
- 贪心：如果用当前字符替换栈顶字符，字典序更小
- 栈顶元素出栈，此时栈顶元素为c
- b还是小于c，且c还会在之后出现，若替换栈顶字符c字典序更小
- 栈顶元素出栈，此时栈顶字符为a，b>a，将b压栈



四. Consider 思考更优解

拉勾教育
—互联网人实战大学—

贪心算法与栈的结合



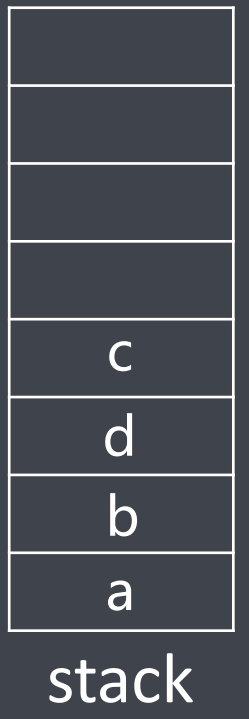
示例

- 当前字符d大于栈顶字符b，压栈
- 当前字符d在栈中出现过，跳过
- 当前字符c小于栈顶字符d，然而d并未在之后出现过

不能替换，压栈

- 剩下的字符都在栈中出现过，返回结果为栈中字符

abdc



stack

五. Code 最优解思路及编码实现

解题思路剖析

- 首先浏览一遍字符串，确定每个字符在串中最后一次出现的位置
- 再浏览一遍字符串，对当前字符：
  - 若栈为空，压栈
  - 若栈不空，且当前字符在栈中出现过，跳过当前字符
  - 否则比较当前字符与栈顶字符
    - 若当前字符大于栈顶字符，压栈
    - 若栈顶字符大于当前字符且最后出现位置在当前位置之后，栈顶出栈，继续比较当前字符与新栈顶字符，直至栈为空或栈顶字符小于当前字符或栈顶字符未在当前位置后出现过，将当前字符压栈

五. Code 最优解思路及编码实现

Java编码实现

```java
public String removeDuplicateLetters(String s) {
    int length = s.length();
    //创建计数器 lastOccurrence 记录每个字母在字符串中最后一次出现的位置
    int[] lastOccurrence = new int[26];
    for (int i = 0; i < length; i++)
        lastOccurrence[s.charAt(i) - 'a'] = i;
    Stack<Character> stack = new Stack<>();
    boolean[] seen = new boolean[26]; //记录当前栈中已经存在的字符
    //再浏览一遍字符串，对当前字符:
    for (int i = 0; i < length; i++) {
        char c = s.charAt(i);
        if (!seen[c - 'a']) {//若栈不空，且当前字符在栈中出现过，跳过当前字符
            while (!stack.isEmpty() && c < stack.peek() &&
lastOccurrence[stack.peek() - 'a'] > i) {//栈为空、或当前字符大于栈顶字符、或栈顶字符在当前字符后不再出现，当前字符压栈
                seen[stack.pop() - 'a'] = false;
            }
            stack.push(c);
            seen[c - 'a'] = true;
        }
    }
    String result = "";
    while (!stack.isEmpty()) result = stack.pop() + result; //栈拼接成字符串
    return result;
}
```

执行结果: 通过 显示详情 >

执行用时: 4 ms , 在所有 Java 提交中击败了 91.26% 的用户

内存消耗: 39.4 MB , 在所有 Java 提交中击败了 9.72% 的用户

五. Code 最优解思路及编码实现

复杂度分析

时间复杂度：

- 统计字符串中出现过的字母在字符串中出现的最后位置，需要对字符串浏览一遍，时间复杂度为O(n)
- 扫描字符串一遍，得到结果，时间复杂度为O(n)
- 总时间复杂度为O(n)

空间复杂度：

- 空间消耗方面我们只需要使用栈来存储结果字符串（最大长度为26）以及其他辅助变量，均为常数级的变量空间，空间复杂度为O(1)

五. Code 最优解思路及编码实现

代码能否进一步优化？

由于栈的后进先出特点，需将栈顶元素拼接到字符串前面

- 考虑使用字符数组来模拟栈，可以左到右顺序地得到字符串
- 同时字符数组为静态空间，可以避免栈的动态扩容
- 考虑使用Java中的StringBuilder来实现拼接

五. Code 最优解思路及编码实现

Java编码实现

执行结果： 通过 显示详情 >

执行用时： 1 ms ，在所有 Java 提交中击败了 100.00% 的用户

内存消耗： 37.1 MB ，在所有 Java 提交中击败了 100.00% 的用户

```java
public String removeDuplicateLetters(String s) {
    //此处使用字符数组来模拟栈，top记录栈顶元素的下标（top=-1时栈为空）
    char[] stack = new char[26];
    int top = -1;
    //数组seen记录当前栈中已经存在的字符，如果后续再遇到可以跳过
    boolean[] seen = new boolean[26];
    //last_occurrence 记录字符串中出现过的字符在字符串最后一次出现的位置
    int[] last_occurrence = new int[26];
    char[] cs = s.toCharArray();
    for(int i = 0; i < s.length(); i++)
        last_occurrence[cs[i]-'a'] = i;
    //从左到右扫描字符串
    for(int i = 0; i < s.length(); i++){
        char c = cs[i];
        if (!seen[c-'a’]){//若当前字符已经在栈中，无需处理
            //如果栈中有元素，且栈顶元素比当前字符小，并且栈顶字符在后续字符串还会出现，那么我们可以用当前字符替换栈顶字符得到一个字典序更小的字符串（注意此处将一直与栈顶元素相比，直到栈为空或栈顶字符比当前字符小，或栈顶字符在当前位置之后不会再出现）
            while(top!=-1 && c < stack[top] && last_occurrence[stack[top]-'a'] > i)
                seen[stack[top--]-'a']=false;
            seen[c-'a'] = true;
            stack[++top]=c;
        }
    }
    //将栈中的字母连接起来
    StringBuilder ss = new StringBuilder();
    for(int i = 0; i <= top; i++)  ss.append(stack[i]);
    return ss.toString();
}
```

六. Change 变形延伸

题目变形

1. 返回字符串 text 中按字典序排列最小的子序列，该子序列包含 text 中所有不同字符一次。
2. 若要求返回字典序最大的字符串应该如何操作？

延伸扩展

总结

- 加深对贪心算法原理的理解
- 巩固在实际问题中对贪心算法的应用
- 结合数据结构栈完成对字符串的处理

课后练习

1. 使括号有效的最少添加（leetcode 921/中等）
2. 不含 AAA 或 BBB 的字符串（leetcode 984/中等）
3. 重构字符串（leetcode 767/中等）
4. 检查字符串是否可以打破另一个字符串(leetcode 1433/中等)

拉勾教育

—互联网人实战大学—

下载「拉勾教育App」
获取更多内容