基本解法

Java代码

```
class Solution {
  public String removeDuplicateLetters(String s) {
        int length = s.length();
        //结束条件：空字符串不处理
        if (length <= 1) return s;
        //主功能：1. 记录每个字母最后出现位置 2.尝试找到尽可能小的首字符 3.在剩余子串中删掉首字符然后递归寻找下一个首字符
        //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;
        }
        // 记录当前处理过的字符串中最小的字符位置
        int pos = 0;
        for (int i = 0; i < length; i++) {
            if (s.charAt(i) < s.charAt(pos)) {
                pos = i;
            }
            //发现某个字符的最后一次出现，那么满足条件的当前字符串的最小字符一定不在后面的子字符串中了，退出循环
            if(lastOccurrence[s.charAt(i)-'a']==i) break;
        }
        //继续处理剩下的pos之后的字符串，我们可以把pos之后字符串中重复出现的字符s.charAt(pos)去除了
        String remainderString=s.substring(pos+1).replace(""+s.charAt(pos),"");
        //关系式：f(n)=x+f(m) x为本轮找到的最小首字符，m剩余子串中剔除首字符得到的子串
        //返回的字符串是最小首字符 + 剩下的字符串去除重复字母后的结果
        return s.charAt(pos)+removeDuplicateLetters(remainderString);
    }
}
```

优化解法

Java代码

```
class Solution {
  public String removeDuplicateLetters(String s) {
    Stack<Character> stack = new Stack<>();
    //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直接跳过
    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];
    for(int i = 0; i < s.length(); i++)
        last_occurrence[s.charAt(i)-'a'] = i;
    //从左到右扫描字符串
    for(int i = 0; i < s.length(); i++){
       char c = s.charAt(i);
       //若当前字符已经在栈中，无需处理
       if (!seen[c-'a']){
         //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（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(!stack.isEmpty() && c < stack.peek() && last_occurrence[stack.peek()-'a'] > i)
               seen[stack.pop()-'a']=false;
           seen[c-'a'] = true;
           stack.push(c);
       }
    }
    String result = "";
    while(!stack.isEmpty())  result = stack.pop() + result;  //将栈中的字母连接起来
    return result;
  }
}
```

最优解法

Java代码

```
class Solution {
  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 result = new StringBuilder();
      for(int i = 0; i <= top; i++)  result.append(stack[i]);
      return result.toString();
  }
}
```

C++代码

```
class Solution {
public:
    string removeDuplicateLetters(string s) {
        string stack;
        for(size_t i = 0;i < s.size(); ++i)
        {
            //当前字符已经在栈中
            if(stack.find(s[i]) != string::npos) continue;
            //只要当前字符比栈顶字符小，且栈顶字符还会出现，弹栈
            while(!stack.empty()&& stack.back() > s[i] && s.find(stack.back(), i) != string::npos)
                stack.pop_back();
            stack.push_back(s[i]);
        }
        return stack;
    }
};
```

Python代码

```
class Solution:
    def removeDuplicateLetters(self, s):
        stack = []
        remain_counter = collections.Counter(s)
        for c in s:
            #当前字符未在栈中，才继续处理
            if c not in stack:
                #只要当前字符比栈顶字符小，且栈顶字符还会出现，弹栈
                while stack and c < stack[-1] and  remain_counter[stack[-1]] >
0:
                    stack.pop()
                stack.append(c)
            remain_counter[c] -= 1
        return ''.join(stack)
```