OpenAI 再次 Close，国内大模型闻声开启抢食大战

OpenAI 再次 Close

近日，国内不少开发者收到来自 OpenAI 官方的通知邮件。

邮件中提到：*OpenAI 将于 7 月 9 日开始限制部分国家/地区的 API 调用，其中包括中国大陆。*

有些读者可能会有疑问，OpenAI 的 ChatGPT 不是一直都不能用吗?

注意：*OpenAI 之前只是主动关闭中国大陆对 ChatGPT 的网页访问，并没有主动关闭 API 访问。而目前，则是采取「额外措施」来终止 API 访问。*

之前没有终止 API 访问，导致国内 AI 套壳工具泛滥，一度成为”热门生意”。

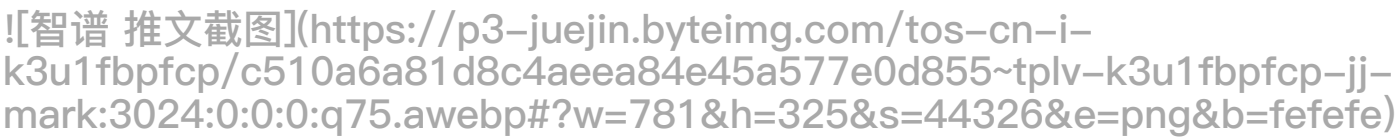
OpenAI 这一”加强服务限制”的决定，将会直接导致这些套壳 AI 原形毕露。

OpenAI 这一决定的意图，我们不得而知，猜想可能与今年 6 月 22 日美国财政部发布的规则草案，要求美国在半导体和微电子、量子计算和人工智能领域的某些投资进行监管密切相关。

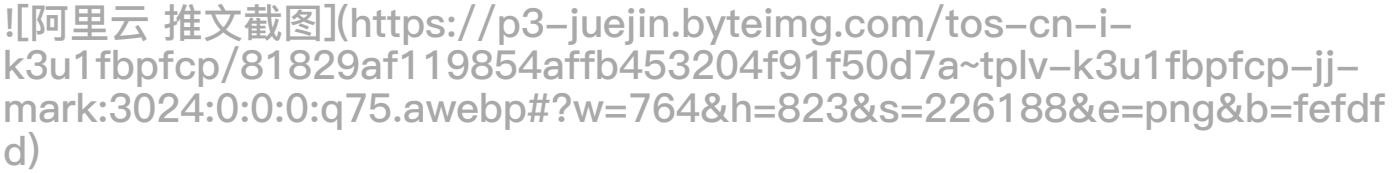
在中国，一门生意的没落，意味着另一门生意的崛起。

在 OpenAI 宣布关停中国 API 访问倒计时的当天，抢食大战也正式打响。

第一家冲跑出来抢占 OpenAI 份额的公司是**智谱 AI**，发文《面向 OpenAI API用户提供特别搬家计划》，高调宣布接手流量：



两个小时后，**阿里云**接力，发文《OpenAI用不了？丝滑迁移通义API!》，表示会为 OpenAI 用户提供最具性价比的中国大模型替代方案：



又过了两个小时，**百度**也出来了，发文《故乡的云·国产大模型普惠计划》，表明为 OpenAI 迁移用户额外赠送使用规模对等的 Tokens 包：

再过了半个小时后，**零一万物**发文《OpenAI 断供中国大陆市场，零一万物 Yi API 二折平替 GPT-4o》，Yi-Large 作为也是国内最能打的选手之一，向用户表示，提供平滑迁移方案，同时强调成本最多可下降九成：

...

嗯，又是熟悉的味道。

卡脖子的高端芯片造不出来，但凡能造出来的东西，分分钟快进到价格战。

...

回归主线。

来一道和「字节跳动」相关的算法原题。

题目描述

平台：LeetCode

题号：899

给定一个字符串 `s` 和一个整数 `k` 。

你可以从 `s` 的前 `k` 个字母中选择一个，并把它加到字符串的末尾。

返回在应用上述步骤的任意数量的移动后，字典上最小的字符串 。

示例 1：

```
输入：s = ”cba”, k = 1

输出：”acb”

解释：
在第一步中，我们将第一个字符（“c”）移动到最后，获得字符串“bac”。
在第二步中，我们将第一个字符（“b”）移动到最后，获得最终结果“acb”。
```

```

示例 2：

```
输入：s = ”baaca”, k = 3

输出：”aaabc”

解释：
在第一步中，我们将第一个字符（“b”）移动到最后，获得字符串“aacab”。
在第二步中，我们将第三个字符（“c”）移动到最后，获得最终结果
“aaabc”。
```

提示：

* 1<=k <=S.length <=10001 <= k <= S.length <=
10001<=k <=S.length <=1000
* `s` 只由小写字母组成。

最小表示法
---

当 k>1k > 1k>1 时，我们能够构造出任意的字符串方案，因此当 k>1k > 1k>1 时，我们可以直接通过对字符串排序来得到答案，复杂度为 O(nlogn)O(n\log{n})O(nlogn)。

当 k=1k = 1k=1 时，我们共有 `n` 种候选方案（将字符串 `s` 看作一个首尾相接的循环字符串，共有 `n` 个起点可枚举），枚举过程中需要与当前最优的方案进行比较，比较复杂度为 O(n)O(n)O(n)，因此整体复杂度为 O(n2)O(n^2)O(n2)。

上述的做法已经可以通过本题，可以看出瓶颈在于对 k=1k = 1k=1 的处理。

而实际上，对于给定字符串 `s`，求其循环同构的所有方案中字典序最小的方案，可以使用「最小表示法」来做，复杂度为 O(n)O(n)O(n)。
```

最小表示法将「方案比较」与「构造更优方案」进行结合：假设我们当前有两字符串 `a` 和 `b` 需要进行比较，其均为原串 `s` 的循环同构具体方案。假设 `a` 和 `b` 分别对应了原串下标为 `i` 和 `j` 的具体方案，且假设两字符串前 `k` 个字符均相同。

当两字符串第一个不同的字符大小关系为 cs[i+k]>cs[j+k]cs[i + k] > cs[j + k]cs[i+k]>cs[j+k] 时，可以发现在下标范围 idx∈[i,i+k]idx \in [i, i + k]idx∈[i,i+k] 作为起点的新方案 `a'` 必然不会是最优方案，即必然存在下标范围 idx∈i+jidx − i + jidx∈i+j 作为起点的新方案 `b'` 比其更优，因此我们可以直接从 i+k+1i + k + 1i+k+1 位置构造新的更优方案，并与 `b` 再次比较。而 cs[i+k]<cs[j+k]cs[i + k] < cs[j + k]cs[i+k]<cs[j+k] 的分析同理。

> 更为直白的表述为：分别从 `i` 和 `j` 作为起点的字符串 `a` 和 `b`，其前 kkk 个字符相同，而当 cs[i+k]>cs[j+k]cs[i + k] > cs[j + k]cs[i+k]>cs[j+k] 时，我们可以明确「以 i+pi + pi+p 为起点的字符串」必不可能比「以 j+pj + pj+p 为起点的字符串」更优，其中 p∈[0,k]p \in [0, k]p∈[0,k]。

Java 代码：

```
class Solution {
    public String orderlyQueue(String s, int _k) {
        char[] cs = s.toCharArray();
        if (_k == 1) {
            int i = 0, j = 1, k = 0, n = cs.length;
            while (i < n && j < n && k < n) {
                char a = cs[(i + k) % n], b = cs[(j + k) % n];
                if (a == b) k++;
                else {
                    if (a > b) i += k + 1;
                    else j += k + 1;
                    if (i == j) i++;
                    k = 0;
                }
            }
            i = Math.min(i, j);
            return s.substring(i) + s.substring(0, i);
        } else {
            Arrays.sort(cs);
            return String.valueOf(cs);
        }
    }
```

}

```

C++ 代码：

```
class Solution {
public:
    string orderlyQueue(string s, int _k) {
        if (_k == 1) {
            int i = 0, j = 1, k = 0, n = s.length();
            while (i < n && j < n && k < n) {
                char a = s[(i + k) % n], b = s[(j + k) % n];
                if (a == b) k++;
                else {
                    if (a > b) i += k + 1;
                    else j += k + 1;
                    if (i == j) i++;
                    k = 0;
                }
            }
            i = min(i, j);
            return s.substr(i) + s.substr(0, i);
        } else {
            char* cs = new char[s.length()];
            copy(s.begin(), s.end(), cs);
            sort(cs, cs + s.length());
            string sortedString(cs, s.length());
            return sortedString;
        }
    }
};
```

Python 代码：

```
class Solution:
    def orderlyQueue(self, s: str, _k: int) -> str:
        if _k == 1:
            i, j, k, n = 0, 1, 0, len(s)
            while i < n and j < n and k < n:
```
```

```
                a, b = s[(i + k) % n], s[(j + k) % n]
                if a == b:
                    k += 1
                else:
                    if a > b:
                        i += k + 1
                    else:
                        j += k + 1
                    if i == j:
                        i += 1
                    k = 0
            i = min(i, j)
            return s[i:] + s[:i]
        else:
            return ''.join(sorted(s))
```

TypeScript 代码：

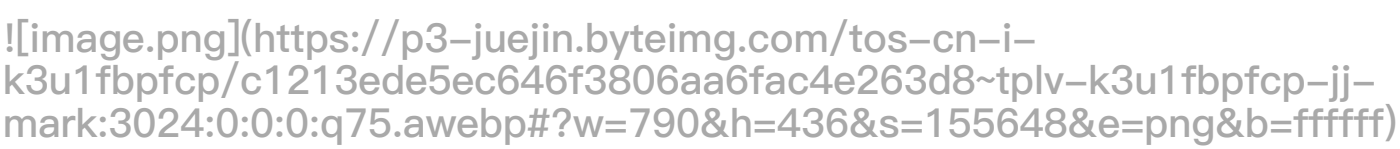
```
function orderlyQueue(s: string, _k: number): string {
    if (_k == 1) {
        let i = 0, j = 1, k = 0, n = s.length
        while (i < n && j < n && k < n) {
            const a = s[(i + k) % n], b = s[(j + k) % n]
            if (a == b) k++;
            else {
                if (a > b) i += k + 1
                else j += k + 1
                if (i == j) i++
                k = 0
            }
        }
        i = Math.min(i, j)
        return s.substring(i) + s.substring(0, i)
    } else {
        return [...s].sort().join('');
    }
};
```

* 时间复杂度：当 k=1k = 1k=1 时，复杂度为 O(n)O(n)O(n)；当 k>1k > 1k>1 时，复杂度为 O(nlogn)O(n\log{n})O(nlogn)
* 空间复杂度：当 k>1k > 1k>1 时，需要使用额外的排序空间

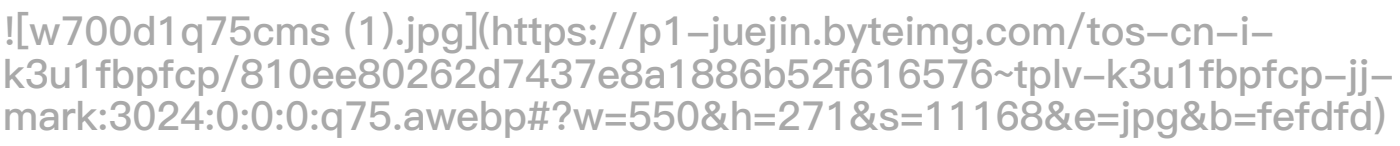
O(logn)O(\log{n})O(logn)

排盲
--

做一些基本的答疑和排盲：



OpenAI 不支持 +86 and +852 手机号注册截图：



原文链接: https://juejin.cn/post/7384750303521456143