08 最长公共前缀

更新时间：2019-08-14 10:07:00

> 学习要注意到细处，不是粗枝大叶的，这样可以逐步学习、摸索，找到客观规律。
>
> —— 徐特立

刷题内容

> 难度：**Easy**
>
> 原题链接：https://leetcode-cn.com/problems/longest-common-prefix/。

内容描述

编写一个函数来查找字符串数组中的最长公共前缀。

如果不存在公共前缀，返回空字符串 ""。

示例 1:
输入: ["flower","flow","flight"]
输出: "fl"

示例 2:
输入: ["dog","racecar","car"]
输出: ""
解释: 输入不存在公共前缀。

说明: 所有输入只包含小写字母 a-z 。

解题方案

思路 1：时间复杂度: O(N * len(strs(0)) 空间复杂度: O(N)

一共有 N 个字符串，题目要求我们求它们的公共前缀（还得是最长的）。那么此时假设我们已经知道前 i+1 个字符串的最长公共前缀（ dp[i] ）了，此时再来一个字符串（即第 i+2 个字符串）。我们需要做什么呢？如果第 i+2 个字符串不以 dp[i] 为前缀，就去掉 dp[i] 的最后一个字符再试一次。假如都去完了呢？那么最后结果就是空串。

下面来看具体的代码：

Python beats 96.61%

```python
class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        if not strs:
            return ''
        dp = [strs[0]] * len(strs)
        for i in range(1, len(strs)):
            # startswith() 方法用于检查字符串是否是以指定子字符串开头
            while not strs[i].startswith(dp[i-1]):
                dp[i-1] = dp[i-1][:-1]
            dp[i] = dp[i-1]
        return dp[-1]
```

Java beats 80.80%

```java
class Solution {
    public String longestCommonPrefix(String[] strs) {
        if (strs.length == 0) {
            return "";
        }
        // 最长字串的长度一定不会超过第 0 个字符串的长度
        int end = strs[0].length();
        for (int i = 1; i < strs.length; i++){
            int j = 0;
            while (j < end && j < strs[i].length() && strs[0].charAt(j) == strs[i].charAt(j)) {
                j++;
            }
            end = j;
        }
        return strs[0].substring(0,end);
    }
}
```

Go beats 91.00%

```go
func longestCommonPrefix(strs []string) string {
 if len(strs) == 0 {
  return ""
 }
 dp := make([]string, len(strs))
 for i := 0; i < len(strs); i += 1 {
  dp[i] = strs[0]
 }
 for i := 1; i < len(strs); i += 1 {
  for !strings.HasPrefix(strs[i], dp[i-1]) {
   dp[i-1] = dp[i-1][:len(dp[i-1])-1]
  }
  dp[i] = dp[i-1]
 }
 return dp[len(dp)-1]
}
```

c++ beats 89.73%

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        if (strs.size() == 0) {
            return "";
        }
        //dp就是strs[0]，所以我们直接用strs[0]，避免内存复制
        int end = strs[0].size();
        for (int i = 1;i < strs.size();i++) {
            int j = 0;
            while (j < end && j < strs[i].size() && strs[0][j] == strs[i][j]) j++;
            end = j;
        }
        return strs[0].substr(0, end);
    }
};
```

这种方法需要我们记录每一次的状态，空间复杂度很高，下面我们在思路2中优化一下。

思路 2：时间复杂度: O(N * len(strs(0)) 空间复杂度: O(1)

在思路 1 中我们还需要记录每一步骤的状态，所以空间复杂度为 `O(N)`，我们能否不要额外空间呢？

以一个小例子来解释，`strs=['laa', 'lab', 'lac']`，如果存在 `LCP（最长公共前缀）` 的话它肯定就在第一个字符串 `strs[0]` 中，并且 `LCP` 的长度肯定不会大于 `strs[0]` 的长度。

- 依次假设 `LCP` 长度为 0 到 `len(strs[0])` ，在每一轮循环中:
  - 只要 `strs` 中存在比当前 `LCP` 长度更短的 `string`，立刻返回上一轮 `LCP`，即 `strs[0][:i]`；
  - 只要 `strs` 中存在当前 `index` 字符与 `LCP` 该 `index` 不相同的字符串，立刻返回上一轮 `LCP`，即`strs[0][:i]`；
- 如果一直没返回，说明 `strs[0]` 本身就是 `LCP`，返回它。

Python beats 72.96%

```
class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        if not strs:
            return ''
        for i in range(len(strs[0])):
            for str in strs:
         # 只要strs中存在比当前长度i更短的string，立刻返回上一轮LCP，即strs[0][:i]
     # 只要strs中存在当前index字符与LCP该index不相同的字符串，立刻返回上一轮LCP，即strs[0][:i]
                if len(str) <= i or strs[0][i] != str[i]:
                    return strs[0][:i]
        return strs[0] # 如果一直没返回，说明strs[0]本身就是LCP，返回它
```

甚至可以一行：

```
class Solution:
    def longestCommonPrefix(self, strs: List[str]) -> str:
        return os.path.commonprefix(strs)
```

Java beats 80.80%

```
class Solution {
    public String longestCommonPrefix(String[] strs) {
        if (strs.length == 0) {
            return "";
        }
        for (int i = 0; i < strs[0].length(); i++) {
            // 第 0 个字符不需要比较，可以直接从第 1 个字符开始进比较
            for (int j = 1; j < strs.length; j++) {
        // 只要strs中存在比当前长度i更短的string，立刻返回上一轮LCP，即strs[0][:i]
    // 只要strs中存在当前index字符与LCP该index不相同的字符串，立刻返回上一轮LCP，即strs[0][:i]
                if (i >= strs[j].length() || strs[0].charAt(i) != strs[j].charAt(i)) {
                    return strs[0].substring(0, i);
                }
            }
        }
        return strs[0]; // 如果一直没返回，说明strs[0]本身就是LCP，返回它
    }
}
```

Go beats 100.00%

```
func longestCommonPrefix(strs []string) string {
 if len(strs) == 0 {
  return ""
 }
 for i := 0; i < len(strs[0]); i += 1 {
        for _, str := range strs {
    // 只要strs中存在比当前长度i更短的string，立刻返回上一轮LCP，即strs[0][:i]
    // 只要strs中存在当前index字符与LCP该index不相同的字符串，立刻返回上一轮LCP，即strs[0][:i]
            if len(str) <= i || strs[0][i] != str[i] {
                return strs[0][:i]
            }
        }
 }
 return strs[0] // 如果一直没返回，说明strs[0]本身就是LCP，返回它
}
```

C++ beats 98.97%

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        if (strs.size() == 0) {
            return "";
        }
        // dp就是strs[0]，所以我们直接用strs[0]，避免内存复制
        for (int i = 0;i < strs[0].size();i++) {
            for (int j = 1;j < strs.size();j++) {
        // 只要strs中存在比当前长度i更短的string，立刻返回上一轮LCP，即strs[0][:i]
    // 只要strs中存在当前index字符与LCP该index不相同的字符串，立刻返回上一轮LCP，即strs[0][:i]
                if (i >= strs[j].size() || strs[0][i] != strs[j][i]) {
                    return strs[0].substr(0, i);
                }
            }
        }
        return strs[0]; // 如果一直没返回，说明strs[0]本身就是LCP，返回它
    }
};
```

思路 2 没有用额外空间，显然这一点比思路 1 要好。

小结

- dp 的解法很好，但是有的时候从独特角度切入，我们甚至可以不需要消耗额外空间；
- 要善于观察，比如思路 2 我们知道公共前缀肯定会存在任何一个 str 中，那么我们甚至可以直接挑出一个最短的 str 来枚举（如果这个 str 真的非常短的话），这样甚至是更省时间，并且不需要额外空间。

}

07 整数转罗马数字&罗马数字转整数

09 有效的括号