最长公共前缀

力扣官方题解 | + 关注
874601 2020.06.14 发布于 未知归属地

官方题解 字符串 二分查找 C++ Go 3+

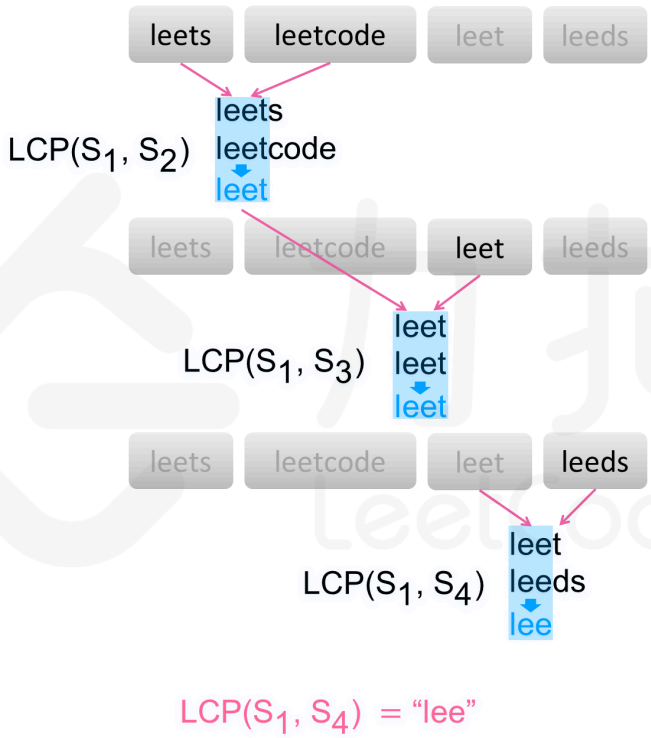
方法一：横向扫描

用 $LCP(S_1 \dots S_n)$ 表示字符串 $S_1 \dots S_n$ 的最长公共前缀。

可以得到以下结论：

$$LCP(S_1 \dots S_n) = LCP(LCP(LCP(S_1, S_2), S_3), \dots S_n)$$

基于该结论，可以得到一种查找字符串数组中的最长公共前缀的简单方法。依次遍历字符串数组中的每个字符串，对于每个遍历到的字符串，更新最长公共前缀，当遍历完所有的字符串以后，即可得到字符串数组中的最长公共前缀。



如果在尚未遍历完所有的字符串时，最长公共前缀已经是空串，则最长公共前缀一定是空串，因此不需要继续遍历剩下的字符串，直接返回空串即可。

Java C++ Python3 Golang

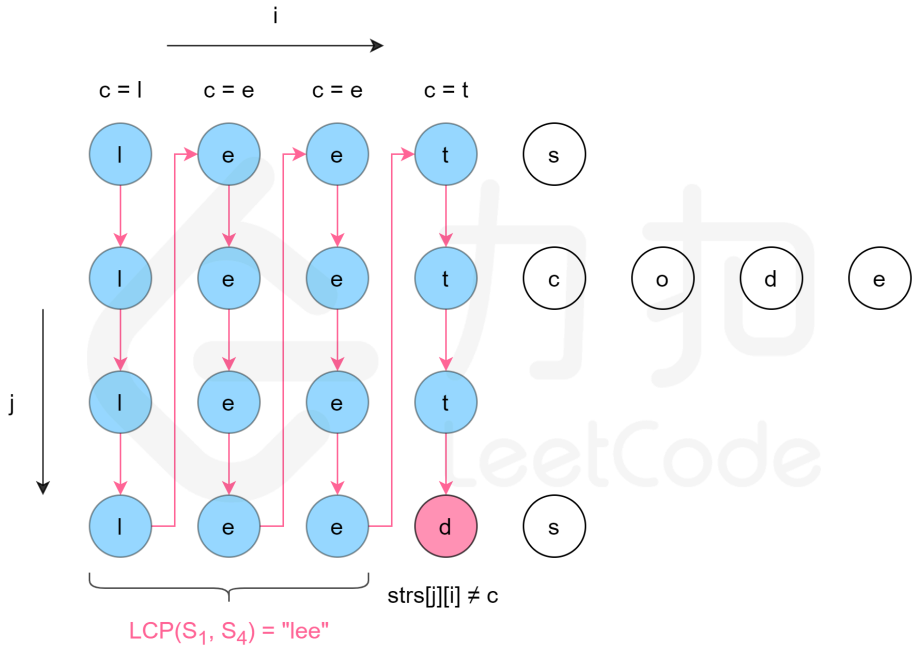
```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        if (!strs.size()) {
            return "";
        }
        string prefix = strs[0];
        int count = strs.size();
        for (int i = 1; i < count; ++i) {
            prefix = longestCommonPrefix(prefix, strs[i]);
            if (!prefix.size()) {
                break;
            }
        }
        return prefix;
    }

    string longestCommonPrefix(const string& str1, const string& str2) {
        int length = min(str1.size(), str2.size());
        int index = 0;
        while (index < length && str1[index] == str2[index]) {
            ++index;
        }
        return str1.substr(0, index);
    }
};
```

复杂度分析

- 时间复杂度：$O(mn)$，其中 m 是字符串数组中的字符串的平均长度，n 是字符串的数量。最坏情况下，字符串数组中的每个字符串的每个字符都会被比较一次。
- 空间复杂度：$O(1)$。使用的额外空间复杂度为常数。

方法二：纵向扫描

方法一是横向扫描，依次遍历每个字符串，更新最长公共前缀。另一种方法是纵向扫描。纵向扫描时，从前往后遍历所有字符串的每一列，比较相同列上的字符是否相同，如果相同则继续对下一列进行比较，如果不相同则当前列不再属于公共前缀，当前列之前的部分为最长公共前缀。



Java | C++ | Python3 | Golang

```cpp
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        if (!strs.size()) {
            return "";
        }
        int length = strs[0].size();
        int count = strs.size();
        for (int i = 0; i < length; ++i) {
            char c = strs[0][i];
            for (int j = 1; j < count; ++j) {
                if (i == strs[j].size() || strs[j][i] != c) {
                    return strs[0].substr(0, i);
                }
            }
        }
        return strs[0];
    }
};
```

复杂度分析

- 时间复杂度：$O(mn)$，其中 m 是字符串数组中的字符串的平均长度，n 是字符串的数量。最坏情况下，字符串数组中的每个字符串的每个字符都会被比较一次。
- 空间复杂度：$O(1)$。使用的额外空间复杂度为常数。

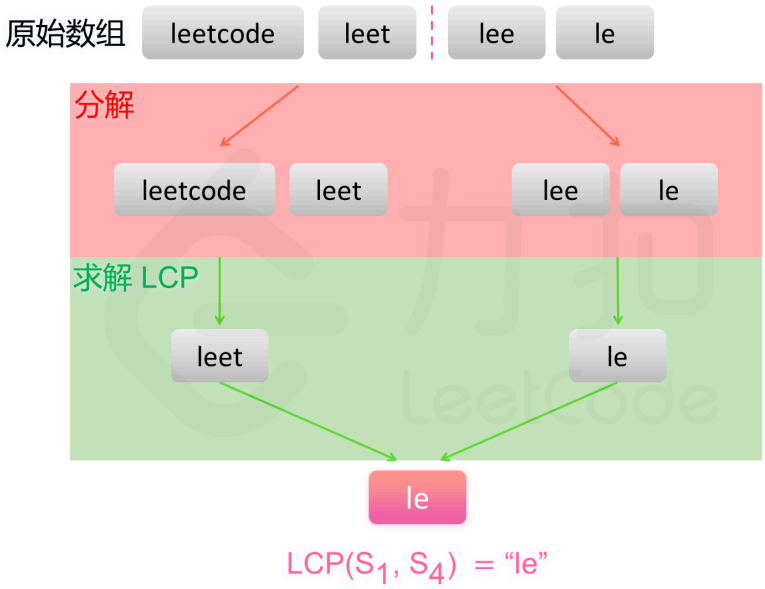
方法三：分治

注意到 LCP 的计算满足结合律，有以下结论：

$$LCP(S_1 \dots S_n) = LCP(LCP(S_1 \dots S_k), LCP(S_{k+1} \dots S_n))$$

其中 $LCP(S_1 \dots S_n)$ 是字符串 $S_1 \dots S_n$ 的最长公共前缀，$1 < k < n$。

基于上述结论，可以使用分治法得到字符串数组中的最长公共前缀。对于问题 $LCP(S_i \cdots S_j)$，可以分解成两个子问题 $LCP(S_i \dots S_{mid})$ 与 $LCP(S_{mid+1} \dots S_j)$，其中 $mid = \frac{i+j}{2}$。对两个子问题分别求解，然后对两个子问题的解计算最长公共前缀，即为原问题的解。



Java | C++ | Python3 | Golang

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        if (!strs.size()) {
            return "";
        }
        else {
```

```
            return longestCommonPrefix(strs, 0, strs.size() - 1);
        }
    }

    string longestCommonPrefix(const vector<string>& strs, int start, int end) {
        if (start == end) {
            return strs[start];
        }
        else {
            int mid = (start + end) / 2;
            string lcpLeft = longestCommonPrefix(strs, start, mid);
            string lcpRight = longestCommonPrefix(strs, mid + 1, end);
            return commonPrefix(lcpLeft, lcpRight);
        }
    }

    string commonPrefix(const string& lcpLeft, const string& lcpRight) {
        int minLength = min(lcpLeft.size(), lcpRight.size());
        for (int i = 0; i < minLength; ++i) {
            if (lcpLeft[i] != lcpRight[i]) {
                return lcpLeft.substr(0, i);
            }
        }
        return lcpLeft.substr(0, minLength);
    }
};
```

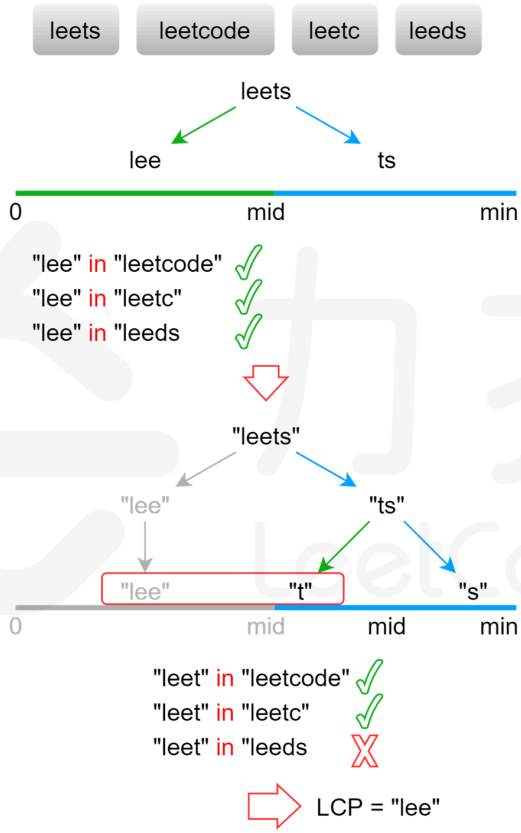
复杂度分析

- 时间复杂度：$O(mn)$，其中 m 是字符串数组中的字符串的平均长度，n 是字符串的数量。时间复杂度的递推式是 $T(n) = 2 \cdot T(\frac{n}{2}) + O(m)$，通过计算可得 $T(n) = O(mn)$。
- 空间复杂度：$O(m \log n)$，其中 m 是字符串数组中的字符串的平均长度，n 是字符串的数量。空间复杂度主要取决于递归调用的层数，层数最大为 $\log n$，每层需要 m 的空间存储返回结果。

方法四：二分查找

显然，最长公共前缀的长度不会超过字符串数组中的最短字符串的长度。用 $minLength$ 表示字符串数组中的最短字符串的长度，则可以在 $[0, minLength]$ 的范围内通过二分查找得到最长公共前缀的长度。每次取查找范围的中间值 mid，判断每个字符串的长度为 mid 的前缀是否相同，如果相同则最

长公共前缀的长度一定大于或等于 mid，如果不相同则最长公共前缀的长度一定小于 mid，通过上述方式将查找范围缩小一半，直到得到最长公共前缀的长度。



Java | C++ | Python3 | Golang

```
class Solution {
public:
    string longestCommonPrefix(vector<string>& strs) {
        if (!strs.size()) {
            return "";
        }
        int minLength = min_element(strs.begin(), strs.end(), [](const string& s, const string&
        int low = 0, high = minLength;
        while (low < high) {
            int mid = (high - low + 1) / 2 + low;
            if (isCommonPrefix(strs, mid)) {
                low = mid;
            }
            else {
                high = mid - 1;
            }
```

```
        }
        return strs[0].substr(0, low);
    }

    bool isCommonPrefix(const vector<string>& strs, int length) {
        string str0 = strs[0].substr(0, length);
        int count = strs.size();
        for (int i = 1; i < count; ++i) {
            string str = strs[i];
            for (int j = 0; j < length; ++j) {
                if (str0[j] != str[j]) {
                    return false;
                }
            }
        }
        return true;
    }
};
```

复杂度分析

- 时间复杂度：$O(mn \log m)$，其中 m 是字符串数组中的字符串的最小长度，n 是字符串的数量。二分查找的迭代执行次数是 $O(\log m)$，每次迭代最多需要比较 mn 个字符，因此总时间复杂度是 $O(mn \log m)$。

- 空间复杂度：$O(1)$。使用的额外空间复杂度为常数。