# 题目

给你一个字符串 s 和一个字符串数组 dictionary ，找出并返回 dictionary 中最长的字符串，该字符串可以通过删除 s 中的某些字符得到。

如果答案不止一个，返回长度最长且字母序最小的字符串。如果答案不存在，则返回空字符串。

示例 1：

输入：s = "abpcplea", dictionary = ["ale","apple","monkey","plea"]

输出："apple"

示例 2：

输入：s = "abpcplea", dictionary = ["a","b","c"]

输出："a"

提示：

1 <= s.length <= 1000

1 <= dictionary.length <= 1000

1 <= dictionary[i].length <= 1000

s 和 dictionary[i] 仅由小写英文字母组成

# 分析

## 方法一：双指针

思路：

以通过遍历dictionary中的每个字符串，然后对每个字符串检查它是否是s的子序列来实现。为了找到最长的且字母序最小的字符串，我们可以维护一个变量来记录当前找到的最长字符串，并在找到更长或相同长度但字母序更小的字符串时更新它。

代码：

class Solution {

public:

string findLongestWord(string s, vector<string>& dictionary) {

// 初始化最长字符串为空字符串

string longestWord = "";

// 遍历dictionary中的每个字符串

for (const string& word : dictionary) {

// 检查当前字符串word是否是s的子序列

if (isSubsequence(s, word) &&

// 并且满足以下条件之一：

// 1. word的长度大于当前记录的最长字符串

// 2. word的长度与当前记录的最长字符串相同，但字母序更小

(word.length() > longestWord.length() ||

(word.length() == longestWord.length() && word < longestWord))) {

// 更新最长字符串

longestWord = word;

}

}

// 返回最长字符串

return longestWord;

}

private:

// 辅助函数：检查一个字符串word是否是另一个字符串s的子序列

bool isSubsequence(const string& s, const string& word) {

// 使用两个指针分别遍历s和word

int i = 0, j = 0;

// 当两个指针都未到达各自字符串的末尾时，进行循环

while (i < s.length() && j < word.length()) {

// 如果当前字符匹配，则将word的指针向前移动

if (s[i] == word[j]) {

j++;

}

// 无论是否匹配，都将s的指针向前移动

i++;

}

// 如果word的指针到达了末尾，说明word是s的子序列

return j == word.length();

}

};

在这个实现中，isSubsequence函数用于检查一个字符串是否是另一个字符串的子序列。它使用两个指针i和j分别遍历s和word，当找到匹配的字符时，将j向前移动。最后，如果j到达了word的末尾，说明word是s的子序列。

在findLongestWord函数中，我们遍历dictionary中的每个字符串，并检查它是否是s的子序列。如果是，并且它的长度大于当前记录的最长字符串longestWord，或者它的长度与longestWord相同但字母序更小，我们就更新longestWord。最后，返回longestWord作为结果。

注意：这个实现的时间复杂度是O(nm)，其中n是dictionary中字符串的数量，m是dictionary中最长字符串的长度。在最坏情况下，我们需要对dictionary中的每个字符串都调用一次isSubsequence函数，而isSubsequence函数的时间复杂度是O(m)。因此，总的时间复杂度是O(nm)。如果dictionary或其中的字符串非常长，这个实现可能会比较慢。在实际应用中，可能需要考虑优化算法或使用更高效的数据结构来减少时间复杂度。

## 方法二：排序

## 方法三：动态规划