# 题目

给你一个下标从0开始的字符串数组words。

定义一个布尔函数isPrefixAndSuffix，它接受两个字符串参数str1和str2：

当str1同时是str2的前缀（prefix）和后缀（suffix）时，isPrefixAndSuffix(str1, str2)返回true，否则返回false。

例如，isPrefixAndSuffix("aba", "ababa")返回true，因为"aba"既是"ababa"的前缀，也是"ababa"的后缀，但是isPrefixAndSuffix("abc", "abcd")返回false。

以整数形式，返回满足i < j且isPrefixAndSuffix(words[i], words[j])为true的下标对(i, j)的数量。

示例 1：

输入：words = ["a","aba","ababa","aa"]

输出：4

解释：在本示例中，计数的下标对包括：

i = 0且j = 1，因为isPrefixAndSuffix("a", "aba")为true。

i = 0且j = 2，因为isPrefixAndSuffix("a", "ababa")为true。

i = 0且j = 3，因为isPrefixAndSuffix("a", "aa")为true。

i = 1且j = 2，因为isPrefixAndSuffix("aba", "ababa")为true。

因此，答案是 4 。

示例 2：

输入：words = ["pa","papa","ma","mama"]

输出：2

解释：在本示例中，计数的下标对包括：

i = 0且j = 1，因为isPrefixAndSuffix("pa", "papa")为true。

i = 2且j = 3，因为isPrefixAndSuffix("ma", "mama")为true。

因此，答案是 2 。

示例 3：

输入：words = ["abab","ab"]

输出：0

解释：在本示例中，唯一有效的下标对是i=0且j=1，但是isPrefixAndSuffix("abab", "ab")为false。

因此，答案是 0 。

提示：

1 <= words.length <= 50

1 <= words[i].length <= 10

words[i] 仅由小写英文字母组成。

# 分析

## 方法一：暴力枚举

class Solution {

public:

bool isPrefixAndSuffix(const string& str1, const string& str2) {

if (str1.size() > str2.size()) {

return false; // str1不能同时是str2的前缀和后缀，如果它的长度大于str2

}

return (str2.substr(0, str1.size()) == str1) && (str2.substr(str2.size() - str1.size()) == str1);

}

int countPrefixSuffixPairs(vector<string>& words) {

int count = 0;

int n = words.size();

// 遍历所有可能的下标对

for (int i = 0; i < n; i++) {

for (int j = i + 1; j < n; j++) {

// 检查当前下标对是否满足条件

if (isPrefixAndSuffix(words[i], words[j])) {

count++;

}

}

}

return count;

}

};

## 方法二：哈希表

思路：

代码：

class Solution {

public:

bool isPrefixAndSuffix(const string& str1, const string& str2) {

if (str1.size() > str2.size()) {

return false; // str1不能同时是str2的前缀和后缀，如果它的长度大于str2

}

return (str2.substr(0, str1.size()) == str1) && (str2.substr(str2.size() - str1.size()) == str1);

}

int countPrefixSuffixPairs(vector<string>& words) {

int count = 0;

unordered\_map<string, vector<int>> wordIndices;

// 构建一个哈希表，将每个单词映射到其在数组中的下标

for (int i = 0; i < words.size(); i++) {

wordIndices[words[i]].push\_back(i);

}

// 遍历每个单词，并检查其与其他单词的关系

for (const auto& entry : wordIndices) {

const string& currentWord = entry.first;

const vector<int>& currentIndices = entry.second;

// 遍历当前单词的所有下标

for (int i : currentIndices) {

// 遍历后续单词，检查是否满足条件

for (int j = i + 1; j < words.size(); j++) {

if (isPrefixAndSuffix(currentWord, words[j])) {

count++;

}

}

}

}

return count;

}

};