# 题目

给你一个字符串数组，请你将 字母异位词 组合在一起。可以按任意顺序返回结果列表。

示例 1:

输入: strs = ["eat", "tea", "tan", "ate", "nat", "bat"]

输出: [["bat"],["nat","tan"],["ate","eat","tea"]]

解释：

在 strs 中没有字符串可以通过重新排列来形成 "bat"。

字符串 "nat" 和 "tan" 是字母异位词，因为它们可以重新排列以形成彼此。

字符串 "ate" ，"eat" 和 "tea" 是字母异位词，因为它们可以重新排列以形成彼此。

示例 2:

输入: strs = [""]

输出: [[""]]

示例 3:

输入: strs = ["a"]

输出: [["a"]]

提示：

1 <= strs.length <= 104

0 <= strs[i].length <= 100

strs[i] 仅包含小写字母

# 分析

要解决将字母异位词组合在一起的问题，我们需要利用字母异位词的核心特性：由相同的字母组成，只是字母顺序不同。因此，我们可以通过为每个字母异位词生成一个唯一的“标识”，将具有相同标识的字符串归类到一起。

解题思路

1、标识生成：对于每个字符串，生成一个唯一标识，使得所有字母异位词的标识相同。常见的方法有两种：

- 排序法：将字符串的字符按字母顺序排序，排序后的字符串作为标识（如 "eat" 和 "tea" 排序后都是 "aet"）。

- 计数法：统计字符串中每个字母的出现次数，用次数序列作为标识（如 "eat" 的标识可表示为 "1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0"，对应26个字母的计数）。

2、分组存储：使用哈希表（`unordered\_map`），以生成的标识为键，以包含该标识所有字符串的列表为值。遍历所有字符串，将每个字符串放入对应的分组中。

3、结果提取：将哈希表中所有的值（即分组后的列表）收集到结果列表中并返回。

## 方法一：排序法

排序法实现简单直观，适合处理字符串长度较短的场景（题目中字符串长度≤100，完全适用）：

代码：

class Solution {

public:

vector<vector<string>> groupAnagrams(vector<string>& strs) {

// 哈希表：键为排序后的字符串（标识），值为字母异位词列表

unordered\_map<string, vector<string>> groups;

for (const string& s : strs) {

// 对字符串排序，生成标识

string key = s;

sort(key.begin(), key.end());

// 将原字符串加入对应分组

groups[key].push\_back(s);

}

// 提取所有分组到结果列表

vector<vector<string>> result;

for (const auto& pair : groups) {

result.push\_back(pair.second);

}

return result;

}

};

解释

核心逻辑：利用排序后字符串的唯一性，将字母异位词映射到同一键。例如 "tan" 和 "nat" 排序后都是 "ant"，因此被分到同一组。

时间复杂度：O(n \* k log k)，其中 n 是字符串数量，k 是字符串的最大长度（排序每个字符串的耗时为 O(k log k)）。

空间复杂度：O(n \* k)，存储所有字符串和哈希表的键。

## 方法二：计数法

计数法避免了排序的时间开销（排序复杂度为O(klog k)，计数为O(k)，其中k是字符串长度），适合对性能要求更高的场景：

class Solution {

public:

vector<vector<string>> groupAnagrams(vector<string>& strs) {

unordered\_map<string, vector<string>> groups;

for (const string& s : strs) {

// 统计每个字母的出现次数（共26个小写字母）

vector<int> count(26, 0);

for (char c : s) {

count[c - 'a']++; // 'a'对应索引0，'b'对应索引1，以此类推

}

// 将计数转换为字符串作为标识（如 "1,0,0,..."）

string key;

for (int i = 0; i < 26; ++i) {

key += to\_string(count[i]) + ","; // 加逗号避免歧义（如11和1,1）

}

// 将原字符串加入对应分组

groups[key].push\_back(s);

}

// 提取结果

vector<vector<string>> result;

for (const auto& pair : groups) {

result.push\_back(pair.second);

}

return result;

}

};

解释：

核心逻辑：通过字母计数生成唯一标识，避免排序开销。例如 "eat" 中 'a'、'e'、't' 各出现1次，计数序列唯一对应这类字母组合。

时间复杂度：O(n \* k)，每个字符串的计数操作耗时 O(k)。

空间复杂度：O(n \* k)，存储所有字符串和哈希表的键（每个键长度固定为26个计数的字符串表示）。