String Similarity Problem

You are given an array words of n strings. Each string has length m and contains only lowercase English letters.

Two strings s and t are similar if we can apply the following operation any number of times (possibly zero times) so that s and t become equal.

Choose either s or t.
Replace every letter in the chosen string with the next letter in the alphabet cyclically. The next letter after 'z' is 'a'.
Count the number of pairs of indices (i, j) such that:

i < j
words[i] and words[j] are similar.
Return an integer denoting the number of such pairs.

Example 1:

Input: words = ["fusion","layout"]

Output: 1

Explanation:

words[0] = "fusion" and words[1] = "layout" are similar because we can apply the operation to "fusion" 6 times. The string "fusion" changes as follows.

"fusion"
"gvtjpo"
"hwukqp"
"ixvlrq"
"jywmsr"
"kzxnts"
"layout"

Example 2:

Input: words = ["ab","aa","za","aa"]

Output: 2

Explanation:

words[0] = "ab" and words[2] = "za" are similar.
words[1] = "aa" and words[3] = "aa" are similar.

Constraints:

1 <= n == words.length <= 105
1 <= m == words[i].length <= 105
1 <= n * m <= 105
words[i] consists only of lowercase English letters.

```
class Solution {
public:
    long long countPairs(vector<string>& words) {

    }
};©leetcode
```

This is a **string normalization / hashing** problem.

Key Insight

The operation:

> Replace every letter in a string with the next letter cyclically (`z → a`)

is equivalent to **adding the same shift (mod 26) to every character**.

So two strings `s` and `t` are **similar iff** there exists a shift `k (0 ≤ k < 26)` such that:

```
rust
```

```
t[i] = (s[i] + k) mod 26   for all i
```

Canonical Form (Normalization)

For each word, build a **canonical representation** that is invariant under cyclic shifts.

How?

For a word `w`:

1. Let `base = w[0]`
2. For each character `w[i]`, compute:

```csharp
normalized[i] = (w[i] - base + 26) % 26
```

3. Store this sequence as a string or vector.

Why this works?

If two words differ only by a global shift, their **relative differences** from the first character will be identical.

Example

Example 1

```
"fusion" → f u s i o n
           0 15 13 3 9 8

"layout" → l a y o u t
           0 15 13 3 9 8
```

Same normalized form → similar ✅

Algorithm

1. Use a hashmap: `map<normalized_form, count>`
2. For each word:
   - Compute its normalized form
   - Increment its count
3. For each group of size `c`, add:

```
r

c * (c - 1) / 2
```

   to the answer.

Time & Space Complexity

- **Time:** `O(n * m)`
- **Space:** `O(n * m)`
- Works within constraints since `n * m ≤ 10^5`

C++ Implementation

```cpp
class Solution {
public:
    long long countPairs(vector<string>& words) {
        unordered_map<string, long long> freq;

        for (const string& w : words) {
            string key;
            int base = w[0] - 'a';

            for (char c : w) {
                int diff = (c - 'a' - base + 26) % 26;
                key.push_back(diff + 'a');  // encode as char
            }

            freq[key]++;
        }
```

```
        long long ans = 0;
        for (auto& [k, c] : freq) {
            ans += c * (c - 1) / 2;
        }

        return ans;
    }
};
```

Example Walkthrough (Example 2)

```
makefile

["ab", "aa", "za", "aa"]

"ab" → "ab"
"za" → "ab"   → pair (0,2)

"aa" → "aa"
"aa" → "aa"   → pair (1,3)

Total = 2
```

Final Notes

- This is a **classic relative-offset hashing problem**
- Same idea appears in problems like *Shifted Strings* on LeetCode
- Efficient and elegant—perfect for large inputs 💡

If you want, I can also show:

- Why comparing all shifts is too slow
- A Python version
- How to debug with sample traces