<https://leetcode.com/problems/longest-palindrome-by-concatenating-two-letter-words/description/>

You are given an array of strings words. Each element of words consists of **two** lowercase English letters.

Create the **longest possible palindrome** by selecting some elements from words and concatenating them in **any order**. Each element can be selected **at most once**.

Return the **length** of the longest palindrome that you can create. If it is impossible to create any palindrome, return 0.

A **palindrome** is a string that reads the same forward and backward.

**Example 1:**

**Input:** words = ["lc","cl","gg"]

**Output:** 6

**Explanation:** One longest palindrome is "lc" + "gg" + "cl" = "lcggcl", of length 6.Note that "clgglc" is another longest palindrome that can be created.

**Example 2:**

**Input:** words = ["ab","ty","yt","lc","cl","ab"]

**Output:** 8

**Explanation:** One longest palindrome is "ty" + "lc" + "cl" + "yt" = "tylcclyt", of length 8.Note that "lcyttycl" is another longest palindrome that can be created.

**Example 3:**

**Input:** words = ["cc","ll","xx"]

**Output:** 2

**Explanation:** One longest palindrome is "cc", of length 2.Note that "ll" is another longest palindrome that can be created, and so is "xx".

**Constraints:**

1 <= words.length <= 10^5

words[i].length == 2

words[i] consists of lowercase English letters.

**Attempt 1: 2025-01-16**

**Solution 1: Hash Table (30 min)**

class Solution {

    public int longestPalindrome(String[] words) {

        int result = 0;

        Map<String, Integer> freq = new HashMap<>();

        for(String w : words) {

            String rev = new StringBuilder(w).reverse().toString();

            // If the reverse word exists in the map, form a pair

            if(freq.getOrDefault(rev, 0) > 0) {

                // Each pair contributes 4 to the palindrome length

                result += 4;

                // Reduce the count of reverse word

                freq.put(rev, freq.get(rev) - 1);

                // Remove if count becomes 0

                if(freq.get(rev) == 0) {

                    freq.remove(rev);

                }

            // Otherwise, add the word to the map

            } else {

                freq.put(w, freq.getOrDefault(w, 0) + 1);

            }

        }

        // Check if there's any unpaired palindromic word like "aa"

        for(String w : freq.keySet()) {

            if(w.charAt(0) == w.charAt(1)) {

                // To form a palindrome, only can add one more remain

                // unpaired palindromic word in the middle of the final

                // string, hence ignore all else ones even same like "aa"

                result += 2;

                break;

            }

        }

        return result;

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to chatGPT**

**Explanation:**

HashMap to Track Word Counts:

Use a HashMap<String, Integer> to store the count of each word.

If a reverse of the current word is found in the map, it forms a palindromic pair.

Handling Reverse Pairs:

When a word’s reverse is already in the map, add 4 to the result (2 for the word and 2 for its reverse).

Decrease the count of the reverse word in the map since it’s used up.

Odd Palindromic Words:

After processing all pairs, check for any leftover palindromic word (like "aa") in the map.

Only one such word can be used as the center of the palindrome, contributing 2 to the result.

Return the Total Length:

Combine the length contributed by paired words and the potential center word.

class Solution {

public int longestPalindrome(String[] words) {

Map<String, Integer> map = new HashMap<>();

int result = 0;

int oddPalindromeCount = 0;

for (String word : words) {

String reverseWord = "" + word.charAt(1) + word.charAt(0); // Reverse the word

if (map.getOrDefault(reverseWord, 0) > 0) {

// If the reverse word exists in the map, form a pair

result += 4; // Each pair contributes 4 to the palindrome length

map.put(reverseWord, map.get(reverseWord) - 1); // Reduce the count of reverse word

if (map.get(reverseWord) == 0) {

map.remove(reverseWord); // Remove if count becomes 0

}

} else {

// Otherwise, add the word to the map

map.put(word, map.getOrDefault(word, 0) + 1);

}

}

// Check if there's any unpaired palindromic word like "aa"

for (String key : map.keySet()) {

if (key.charAt(0) == key.charAt(1) && map.get(key) > 0) {

oddPalindromeCount = 2; // Only one such word can contribute to the center

break;

}

}

return result + oddPalindromeCount;

}

}

**omplexity:**

Time Complexity:

O(n), where n is the number of words in the array. Each word is processed once.

Space Complexity:

O(m), where m is the number of unique words in the input. The HashMap stores counts of unique words.

**Refer to**

<https://leetcode.com/problems/longest-palindrome-by-concatenating-two-letter-words/solutions/1675343/python3-java-c-counting-mirror-words-o-n/>

**Counting Mirror Words O(n)**

***Explanation:***

2 letter words can be of 2 types:

Where both letters are same

Where both letters are different

Based on the above information:

If we are able to find the mirror of a word, ans += 4

The variable **unpaired** is used to store the number of unpaired words with both letters same.

Unpaired here means a word that has not found its mirror word.

At the end if unpaired same letter words are > 0, we can use one of them as the center of the palindromic string.

**Solution 1: With HashMap**

public int longestPalindrome(String[] words) {

HashMap<String, Integer> m = new HashMap();

int unpaired = 0, ans = 0;

for (String w: words) {

if (!m.containsKey(w)) m.put(w, 0);

if (w.charAt(0) == w.charAt(1)) {

if (m.get(w) > 0) {

unpaired--;

m.put(w, m.get(w) - 1);

ans += 4;

}

else {

m.put(w, m.get(w) + 1);

unpaired++;

}

}

else {

String rev = Character.toString(w.charAt(1)) +

Character.toString(w.charAt(0));

if (m.containsKey(rev) && m.get(rev) > 0) {

ans += 4;

m.put(rev, m.get(rev) - 1);

}

else m.put(w, m.get(w) + 1);

}

}

if (unpaired > 0) ans += 2;

return ans;

}

**Solution 2: Without HashMap**

public int longestPalindrome(String[] words) {

int counter[][] = new int[26][26];

int ans = 0;

for (String w: words) {

int a = w.charAt(0) - 'a', b = w.charAt(1) - 'a';

if (counter[b][a] > 0) {

ans += 4;

counter[b][a]--;

}

else counter[a][b]++;

}

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

if (counter[i][i] > 0) {

ans += 2;

break;

}

}

return ans;

}

**Refer to**

<https://leetcode.com/problems/longest-palindrome-by-concatenating-two-letter-words/solutions/2772399/c-without-hashmap-using-2d-counter-array/>

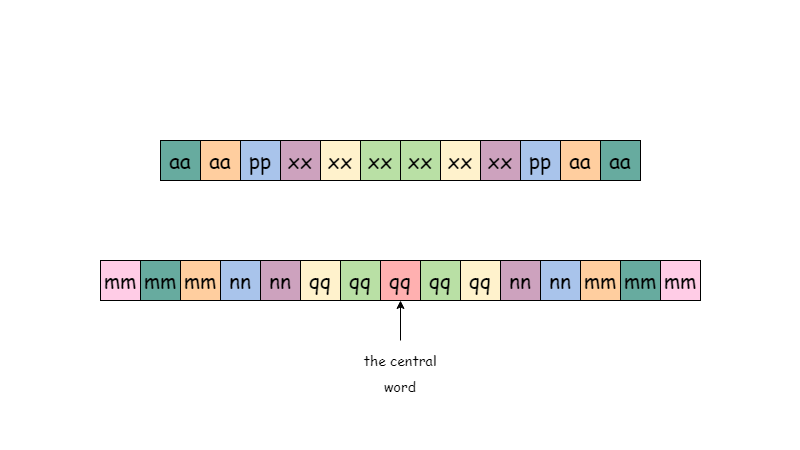
Intuition:

We are given a words array which contains only strings each of Two Letters.

So, we basically have two conditions to check:

When both letters are different

When both letters are same



class Solution {

public:

int longestPalindrome(vector<string>& words) {

int n = words.size();

vector<vector<int>> counter(26,vector<int>(26,0));

int ans=0;

// for different letters in the word

for(string s:words){

int a = s[0]-'a'; // first letter

int b = s[1]-'a'; // second letter

// if the reverse of the word exists i.e like for "lc" if "cl" exists

if(counter[b][a]) {

ans+=4; // count increase by 2+2 = 4

counter[b][a]--; // remove the occurance of the word from counter

}

else counter[a][b]++; // if original doesn't exits in counter array then increase in counter

}

// for same letters in the word

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

if(counter[i][i]){ // if both the letters are same

ans+=2; // increase by 2 i.e like for "gg"

break;

}

}

return ans;

}

};

**Refer to**

[L336.Palindrome Pairs (Ref.L5,L214)](note://883D658FFA6E4B7587F7F47404965C31)

[L409.Longest Palindrome](note://WEBbbb08e20af8e0017b901613f1741f6da)