<https://leetcode.com/problems/remove-letter-to-equalize-frequency/description/>

You are given a **0-indexed** string word, consisting of lowercase English letters. You need to select **one** index and **remove** the letter at that index from word so that the **frequency** of every letter present in word is equal.

Returntrue**if it is possible to remove one letter so that the frequency of all letters in**word**are equal, and**false**otherwise**.

**Note:**

The **frequency** of a letter x is the number of times it occurs in the string.

You **must** remove exactly one letter and cannot choose to do nothing.

**Example 1:**

**Input:** word = "abcc"

**Output:** true

**Explanation:** Select index 3 and delete it: word becomes "abc" and each character has a frequency of 1.

**Example 2:**

**Input:** word = "aazz"

**Output:** false

**Explanation:** We must delete a character, so either the frequency of "a" is 1 and the frequency of "z" is 2, or vice versa. It is impossible to make all present letters have equal frequency.

**Constraints:**

2 <= word.length <= 100

word consists of lowercase English letters only

**Attempt 1: 2024-01-24**

**Solution 1: Hash Table (180 min)**

**难点：各种edge case**

Condition 1: All frequency = 1 except one has 2 (e.g "abcc")

Condition 2: All frequency % 2 = 0 except one % 2 = 1 and this one frequency = 1 (e.g "aab", after remove the only one 'b', this char totally gone, if Equalize Frequency stands, the remain chars only need to consider themselves frequencies)

Condition 3: All frequency % 2 = 0 except one % 2 = 1 and this one frequency > 1 (e.g "aaaaabb", after remove one 'a', this char still exists, its frequency -1, if then the new frequency after remove has to be same as remain chars)

Condition 4: All frequency = 1, may include multiple frequency 0 (e.g "bac")

Condition 5: All frequency = 0 except one (e.g "zzz...zz" 100 'z')

Condition 6: All exist frequency equal(not 0 ones) except one frequency = 1 (e.g "cccd")

**Correct Solution**

class Solution {

    public boolean equalFrequency(String word) {

        int[] freq = new int[26];

        for(char c : word.toCharArray()) {

            freq[c - 'a']++;

        }

        int odd\_count = 0;

        int even\_count = 0;

        int freq\_one\_count = 0;

        int freq\_two\_count = 0;

        int freq\_other\_count = 0;

        int freq\_zero\_count = 0;

        for(int i = 0; i < 26; i++) {

            if(freq[i] % 2 == 0) {

                even\_count++;

            } else {

                odd\_count++;

            }

            if(freq[i] == 1) {

                freq\_one\_count++;

            } else if(freq[i] == 2) {

                freq\_two\_count++;

            } else if(freq[i] == 0){

                freq\_zero\_count++;

            }

        }

        // Condition 1: All frequency = 1 except one has 2 (e.g "abcc")

        // Condition 2: All frequency % 2 = 0 except one % 2 = 1 and this one frequency = 1 (e.g "aab", after remove the only one 'b', this char totally gone, if Equalize Frequency stands, the remain chars only need to consider themselves frequencies)

        // Condition 3: All frequency % 2 = 0 except one % 2 = 1 and this one frequency > 1 (e.g "aaaaabb", after remove one 'a', this char still exists, its frequency -1, if  then the new frequency after remove has to be same as remain chars)

        // Condition 4: All frequency = 1, may include multiple frequency 0 (e.g "bac")

        // Condition 5: All frequency = 0 except one (e.g "zzz...zz" 100 'z')

        // Condition 6: All exist frequency equal(not 0 ones) except one frequency = 1 (e.g "cccd")

        // Handle Condition 2 & 3

        if(odd\_count == 1) {

            int odd\_f = 1;

            int odd\_index = 0;

            int f = 0;

            boolean first\_time = true;

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

                if(freq[i] % 2 == 1 && freq[i] != 1) {

                    odd\_f = freq[i];

                    odd\_index = i;

                    break;

                }

            }

            // Find expected even frequency 'f'

            // Note: During the process to find the even frequency 'f',

            // we must skip the check with the only odd frequency 'odd\_f',

            // because that 'odd\_f' may create confusion result when it

            // not equal to 'f', the easiest way to exclude check on the

            // only odd frequency is skip its index as 'odd\_index' which

            // recorded earlier when we find 'odd\_f'

            for(int i = 0; i < 26 && i != odd\_index; i++) {

                if(!first\_time) {

                    if(freq[i] != 0 && freq[i] != 1 && freq[i] != f) {

                        return false;

                    }

                } else if(freq[i] != 0 && freq[i] != 1) {

                    f = freq[i];

                    first\_time = false;

                }

            }

            // After all, if go with Condtion 3 as odd\_f > 1, to Equalize Frequency

            // stands, then 'odd\_f - 1' must equal to 'f' as we only remove 1 char

            // in the only one odd frequency char, otherwise not stands

            if(odd\_f != 1) {

                if(odd\_f - 1 != f) {

                    return false;

                }

            }

            return true;

        } else {

            // Handle Condition 1

            if(freq\_two\_count == 1) {

                if(freq\_one\_count + freq\_zero\_count == 25) {

                    return true;

                }

            // Handle Condition 4 & 5

            } else if(freq\_one\_count + freq\_zero\_count == 26 || freq\_zero\_count == 25) {

                return true;

            // Handle Condition 6

            } else if(freq\_one\_count == 1) {

                int f = 0;

                boolean first\_time = true;

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

                    if(!first\_time) {

                        if(freq[i] != 0 && freq[i] != 1 && freq[i] != f) {

                            return false;

                        }

                    } else if(freq[i] != 0 && freq[i] != 1) {

                        f = freq[i];

                        first\_time = false;

                    }

                }

                return true;

            }

        }

        return false;

    }

}

Time Complexity: O(N)

Space Complexity: O(N)