<https://leetcode.com/problems/odd-string-difference/description/>

You are given an array of equal-length strings words. Assume that the length of each string is n.

Each string words[i] can be converted into a **difference integer array** difference[i] of length n - 1 where difference[i][j] = words[i][j+1] - words[i][j] where 0 <= j <= n - 2. Note that the difference between two letters is the difference between their **positions** in the alphabet i.e. the position of 'a' is 0, 'b' is 1, and 'z' is 25.

For example, for the string "acb", the difference integer array is [2 - 0, 1 - 2] = [2, -1].

All the strings in words have the same difference integer array, **except one**. You should find that string.

Return*the string in*words*that has different****difference integer array****.*

**Example 1:**

**Input:** words = ["adc","wzy","abc"]

**Output:** "abc"

**Explanation:**

- The difference integer array of "adc" is [3 - 0, 2 - 3] = [3, -1].

- The difference integer array of "wzy" is [25 - 22, 24 - 25]= [3, -1].

- The difference integer array of "abc" is [1 - 0, 2 - 1] = [1, 1].

The odd array out is [1, 1], so we return the corresponding string, "abc".

**Example 2:**

**Input:** words = ["aaa","bob","ccc","ddd"]

**Output:** "bob"

**Explanation:** All the integer arrays are [0, 0] except for "bob", which corresponds to [13, -13].

**Constraints:**

3 <= words.length <= 100

n == words[i].length

2 <= n <= 20

words[i] consists of lowercase English letters.

**Attempt 1: 2025-06-01**

**Solution 1: Hash Table (10 min)**

class Solution {

    public String oddString(String[] words) {

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

        for(String word : words) {

            String normalized = normalizeString(word);

            map.putIfAbsent(normalized, new ArrayList<>());

            map.get(normalized).add(word);

        }

        for(List<String> list : map.values()) {

            if(list.size() == 1) {

                return list.get(0);

            }

        }

        return "";

    }

    private String normalizeString(String word) {

        StringBuilder sb = new StringBuilder();

        sb.append("0");

        char c = word.charAt(0);

        for(int i = 1; i < word.length(); i++) {

            // No worry about if difference is negative value

            // since we store as string

            sb.append(String.valueOf(word.charAt(i) - c));

        }

        return sb.toString();

    }

}

Time Complexity: O(n\*m) where n is the number of words and m is the average length of the words

Space Complexity: O(n)

**Refer to Deepseek**

**LeetCode 2451. Odd String Difference - Java Solution**

This problem requires us to find the string in an array that has a different difference array compared to all others. Here's a Java solution:

import java.util.Arrays;

class Solution {

public String oddString(String[] words) {

// Handle edge case where there are only two words (one must be different)

if (words.length == 2) {

return words[0];

}

// Get the difference array of the first three words

String diff0 = getDifferenceString(words[0]);

String diff1 = getDifferenceString(words[1]);

String diff2 = getDifferenceString(words[2]);

// Determine which pattern is the common one

String commonPattern;

if (diff0.equals(diff1)) {

commonPattern = diff0;

} else if (diff0.equals(diff2)) {

commonPattern = diff0;

} else {

commonPattern = diff1;

}

// Find the word that doesn't match the common pattern

for (String word : words) {

if (!getDifferenceString(word).equals(commonPattern)) {

return word;

}

}

// This line should theoretically never be reached given problem constraints

return "";

}

private String getDifferenceString(String word) {

StringBuilder sb = new StringBuilder();

for (int i = 1; i < word.length(); i++) {

int diff = word.charAt(i) - word.charAt(i - 1);

sb.append(diff).append(",");

}

return sb.toString();

}

}

**Explanation:**

**Edge Case Handling**: If there are only 2 words, one must be different, so we can return the first one (or second, doesn't matter as we'll check patterns later).

**Pattern Identification**: We compare the difference arrays of the first three words to identify the common pattern. The odd one out will be different from the other two.

**Difference Calculation**: The helper method getDifferenceString calculates the difference between each consecutive character in the word and returns it as a string (e.g., "abc" becomes "1,1,").

**Finding the Odd String**: We then iterate through all words to find the one whose difference string doesn't match the common pattern.

This solution efficiently identifies the odd string by first determining the common pattern among the majority of strings and then finding the one that doesn't match. The time complexity is O(n\*m) where n is the number of words and m is the average length of the words.

**How about my solution ?**

Your solution is actually quite elegant and efficient! Let me analyze it and compare it with the previous solution:

**Analysis of Your Solution:**

**Approach**:

You're using a HashMap to group words by their "normalized" difference pattern.

The normalization starts with "0" (the first character's difference with itself) followed by differences between subsequent characters and the first character.

After building the map, you simply look for the entry with only one string, which must be the odd one out.

**Advantages**:

More concise and cleaner implementation

Handles all cases (including edge cases) naturally

Uses a single pass to build the map and another pass to find the odd string

The normalization approach is clever and works well for this problem

**Potential Considerations**:

Your normalization is slightly different from the standard difference array (which typically compares consecutive characters)

However, this doesn't affect correctness since all strings in the same group will have consistent normalization

The "0" prefix isn't strictly necessary but doesn't hurt

**Comparison with Previous Solution:**

**Difference in Normalization:**

**Previous solution: differences between consecutive characters (e.g., "abc" → "1,1,")**

**Your solution: differences from first character (e.g., "abc" → "01,2,")**

**Performance**:

Both solutions are O(n\*m) time complexity (n words, average length m)

Your solution might use slightly more memory due to storing all strings in the map

Both solutions are efficient enough for the problem constraints

**Correctness**:

Both solutions correctly identify the odd string

Your solution handles edge cases more elegantly

**Verdict:**

Your solution is actually better than the initial one I provided! It's:

More concise

More elegant in its approach

Handles edge cases more naturally

Equally efficient in terms of time complexity

The only very minor consideration is that your normalization is slightly different from what might be considered the "standard" difference array, but this doesn't affect correctness and is actually a clever approach.

Great job! Your solution would be perfectly acceptable for LeetCode and demonstrates good problem-solving skills.

**Refer to**

[L2244.Minimum Rounds to Complete All Tasks (Ref.L2451)](note://WEB836f01dee30b51b6bf5f6e87afbd8795)