<https://leetcode.com/problems/count-the-number-of-consistent-strings/description/>

You are given a string allowed consisting of **distinct** characters and an array of strings words. A string is **consistent**if all characters in the string appear in the string allowed.

Return*the number of****consistent****strings in the array*words.

**Example 1:**

**Input:** allowed = "ab", words = ["ad","bd","aaab","baa","badab"]

**Output:** 2

**Explanation:** Strings "aaab" and "baa" are consistent since they only contain characters 'a' and 'b'.

**Example 2:**

**Input:** allowed = "abc", words = ["a","b","c","ab","ac","bc","abc"]

**Output:** 7

**Explanation:** All strings are consistent.

**Example 3:**

**Input:** allowed = "cad", words = ["cc","acd","b","ba","bac","bad","ac","d"]

**Output:** 4

**Explanation:** Strings "cc", "acd", "ac", and "d" are consistent.

**Constraints:**

1 <= words.length <= 10^4

1 <= allowed.length <= 26

1 <= words[i].length <= 10

The characters in allowed are **distinct**.

words[i] and allowed contain only lowercase English letters

**Attempt 1: 2025-03-31**

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

class Solution {

    public int countConsistentStrings(String allowed, String[] words) {

        Set<Character> set = new HashSet<>();

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

            set.add(c);

        }

        int count = 0;

        for(String word : words) {

            if(isConsistent(word, set)) {

                count++;

            }

        }

        return count;

    }

    private boolean isConsistent(String word, Set<Character> set) {

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

            if(!set.contains(c)) {

                return false;

            }

        }

        return true;

    }

}

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

Space Complexity: O(1) for the allowed set (fixed size since only lowercase English letters)

**Solution 2: Bit Manipulation (30 min)**

class Solution {

    public int countConsistentStrings(String allowed, String[] words) {

        int mask = 0;

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

            // Mask Creation: Each bit in an integer represents whether

            // a character is allowed

            // 1 << (c - 'a') sets the bit corresponding to the character's position

            // |= combines all allowed characters into one mask

            mask |= 1 << (c - 'a');

        }

        int count = 0;

        for(String word : words) {

            if(isConsistent(word, mask)) {

                count++;

            }

        }

        return count;

    }

    private boolean isConsistent(String word, int mask) {

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

            // Check if the bit is not set

            if((mask & (1 << (w - 'a'))) == 0) {

                return false;

            }

        }

        return true;

    }

}

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

Space Complexity: O(1) for the allowed set (fixed size since only lowercase English letters)

**Refer to Deepseek**

This problem requires counting how many strings in a given array are "consistent" - meaning they only contain characters from an allowed set.

**Approach**

Create Allowed Character Set: Convert the allowed characters into a set for O(1) lookups

Check Each String: For each string in the input array, verify if all its characters are in the allowed set

Count Consistent Strings: Maintain a count of strings that meet the consistency criteria

**Solution Code**

class Solution {

public int countConsistentStrings(String allowed, String[] words) {

Set<Character> allowedSet = new HashSet<>();

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

allowedSet.add(c);

}

int count = 0;

for (String word : words) {

if (isConsistent(word, allowedSet)) {

count++;

}

}

return count;

}

private boolean isConsistent(String word, Set<Character> allowedSet) {

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

if (!allowedSet.contains(c)) {

return false;

}

}

return true;

}

}

**Explanation**

Allowed Set Creation: We first convert the allowed string into a HashSet for O(1) character lookups

Consistency Check: For each word, we check if all its characters exist in the allowed set

Counting: We increment our count for each word that passes the consistency check

**Complexity Analysis**

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

Space Complexity: O(1) for the allowed set (fixed size since only lowercase English letters)

**Optimized Solution (Using Bitmask)**

For better performance, we can use a bitmask to represent allowed characters:

class Solution {

public int countConsistentStrings(String allowed, String[] words) {

int mask = 0;

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

mask |= 1 << (c - 'a');

}

int count = 0;

for (String word : words) {

if (isConsistent(word, mask)) {

count++;

}

}

return count;

}

private boolean isConsistent(String word, int mask) {

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

if ((mask & (1 << (c - 'a'))) == 0) {

return false;

}

}

return true;

}

}

**Bitmask Explanation**

Mask Creation: Each bit in an integer represents whether a character is allowed

1 << (c - 'a') sets the bit corresponding to the character's position

|= combines all allowed characters into one mask

Consistency Check: For each character in a word, we check if its corresponding bit is set in the mask

(mask & (1 << (c - 'a'))) == 0 checks if the bit is not set

This approach reduces space usage and can be faster due to bitwise operations being very efficient.

**Refer to**

[L2506.Count Pairs Of Similar Strings (Ref.L451)](note://WEB3e9b6b9eba3c0e5a162c03afb590c70b)