<https://leetcode.com/problems/count-vowel-substrings-of-a-string/description/>

A substring is a contiguous (non-empty) sequence of characters within a string.

A vowel substring is a substring that only consists of vowels ('a', 'e', 'i', 'o', and 'u') and has all five vowels present in it.

Given a string word, return the number of vowel substrings in word.

**Example 1:**

**Input:** word = "aeiouu"

**Output:** 2

**Explanation:** The vowel substrings of word are as follows (underlined):

- "**aeiou**u"

- "**aeiouu**"

**Example 2:**

**Input:** word = "unicornarihan"

**Output:** 0

**Explanation:** Not all 5 vowels are present, so there are no vowel substrings.

**Example 3:**

**Input:** word = "cuaieuouac"

**Output:** 7

**Explanation:** The vowel substrings of word are as follows (underlined):

- "c**uaieuo**uac"

- "c**uaieuou**ac"

- "c**uaieuoua**c"

- "cu**aieuo**uac"

- "cu**aieuou**ac"

- "cu**aieuoua**c"

- "cua**ieuoua**c"

**Constraints:**

1 <= word.length <= 100

word consists of lowercase English letters only.

**Attempt 1: 2024-12-30**

**This problem tag as Easy but actually a Hard problem**

**Solution 1: Brute force + HashSet (60 min)**

**Style 1: isVowel() + vowel\_count**

class Solution {

    public int countVowelSubstrings(String word) {

        int count = 0;

        for(int i = 0; i < word.length(); i++) {

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

            int vowel\_count = 0;

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

                char c = word.charAt(j);

                if(isVowel(c)) {

                    if(set.add(c)) {

                        vowel\_count++;

                    }

                } else {

                    break;

                }

                if(vowel\_count == 5) {

                    count++;

                }

            }

        }

        return count;

    }

    private boolean isVowel(char c) {

        return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';

    }

}

Time Complexity: O(n^2)

Space Complexity: O(n)

**Style 2: ! isVowel() + vowel\_count**

class Solution {

    public int countVowelSubstrings(String word) {

        int count = 0;

        for(int i = 0; i < word.length(); i++) {

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

            int vowel\_count = 0;

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

                char c = word.charAt(j);

                if(!isVowel(c)) {

                    break;

                }

                if(set.add(c)) {

                    vowel\_count++;

                }

                if(vowel\_count == 5) {

                    count++;

                }

            }

        }

        return count;

    }

    private boolean isVowel(char c) {

        return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';

    }

}

Time Complexity: O(n^2)

Space Complexity: O(n)

**Style 3: No need vowel\_count**

class Solution {

    public int countVowelSubstrings(String word) {

        int count = 0;

        for(int i = 0; i < word.length(); i++) {

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

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

                char c = word.charAt(j);

                if(!isVowel(c)) {

                    break;

                }

                set.add(c);

                if(set.size() == 5) {

                    count++;

                }

            }

        }

        return count;

    }

    private boolean isVowel(char c) {

        return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';

    }

}

Time Complexity: O(n^2)

Space Complexity: O(n)

**Refer to**

<https://leetcode.com/problems/count-vowel-substrings-of-a-string/solutions/1563906/java-brute-force-hashset-8ms-easy-to-understand/>

Approach:

Two loops are used to find every possible substring starting at i=0 to i = (length)-4 because the minimum length of substring must be 5 to contain each of the vowels.

A HashSet maintains the vowels encountered in each substring which is cleared after each iteration.

If the size of the set is 5, we know that each of the 5 vowels are present.

If a consonant is found, we break out of the inner loop.

class Solution {

public int countVowelSubstrings(String word) {

int vow = 0;

int n = word.length();

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

for(int i = 0; i < n-4; i++) {

set.clear();

for(int j = i; j < n; j++) {

char ch = word.charAt(j);

if(ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u') {

set.add(ch);

if(set.size() == 5)

vow++;

} else {

break;

}

}

}

return vow;

}

}

**Refer to chatGPT**

class Solution {

public int countVowelSubstrings(String word) {

int count = 0;

// Iterate over the string to find all substrings

for (int i = 0; i < word.length(); i++) {

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

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

char c = word.charAt(j);

// If the character is not a vowel, break the inner loop

if (!isVowel(c)) {

break;

}

// Add the vowel to the set

vowels.add(c);

// If the set contains all 5 vowels, increment the count

if (vowels.size() == 5) {

count++;

}

}

}

return count;

}

// Helper method to check if a character is a vowel

private boolean isVowel(char c) {

return c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u';

}

}

**Explanation**

Outer Loop:

Start from each character in the string (i).

The goal is to find all possible substrings starting from index i.

Inner Loop:

Continue iterating through the substring from index i to the end of the string (j).

Check if the current character is a vowel:

**If not a vowel, break the loop since it can no longer form a vowel substring.**

If it is a vowel, add it to a Set<Character> to track distinct vowels.

If the Set contains all 5 vowels (a, e, i, o, u), increment the count.

Helper Method:

The isVowel method determines if a character is a vowel for easy reuse.

**Solution 2: Not fixed length Sliding Window + HashMap (60 min)**

class Solution {

    public int countVowelSubstrings(String word) {

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

        map.put('a', 0);

        map.put('e', 0);

        map.put('i', 0);

        map.put('o', 0);

        map.put('u', 0);

        int subarray\_count = 0;

        int vowel\_count = 0;

        int k = 0;

        int i = 0;

        for(int j = 0; j < word.length(); j++) {

            char c = word.charAt(j);

            if(map.get(c) != null) {

                map.put(c, map.get(c) + 1);

                // Only the first time happen of 'a', 'e', 'i', 'o', and 'u'

                // will contribute to the vowel count

                if(map.get(c) == 1) {

                    vowel\_count++;

                }

                // We cannot directly add (j - i) on total subarray count,

                // since window [i, j] only guarantee when loop till right

                // boundary j, its first time we collect 5 vowels, but left

                // boundary i not guarantee the window [i, j] is the minimum

                // window contains 5 vowels, we have to shrink left boundary

                // to see what's the minimum window contains 5 vowels, and

                // this requires new pointer k start from left(we keep i not

                // touched at this moment to tag the original left boundary),

                // if we get a range [i, k] which plus any character in

                // [k + 1, j] will contains 5 vowels, where i <= k < j, then

                // (k - i + 1) is the valid subarray count which contains

                // 5 vowels and should add into total count

                while(vowel\_count == 5) {

                    char c1 = word.charAt(k);

                    map.put(c1, map.get(c1) - 1);

                    if(map.get(c1) == 0) {

                        vowel\_count--;

                    }

                    k++;

                }

                // No '+ 1' required in detail implementation since

                // '+ 1' implicitly done during last while loop

                subarray\_count += (k - i);

            } else {

                map.forEach((key, value) -> {map.put(key, 0);});

                vowel\_count = 0;

                i = j + 1;

                k = j + 1;

            }

        }

        return subarray\_count;

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**Refer to**

<https://leetcode.com/problems/count-vowel-substrings-of-a-string/solutions/1563737/sliding-window/>

<https://leetcode.com/problems/count-vowel-substrings-of-a-string/solutions/1563737/sliding-window/comments/1141505>

I spent more time on this problem than on medium/hard problems in the contest.

Brute-force would do for 100 characters (since it's an easy problem), but I wanted to solve it efficiently.

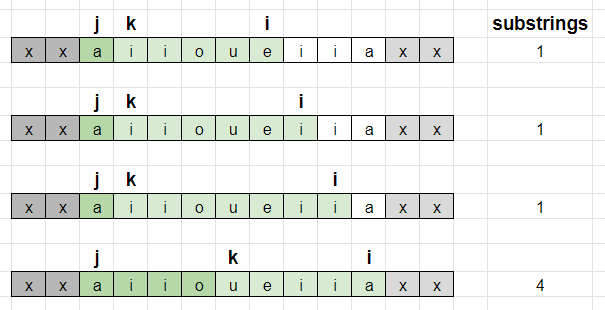
j marks the start of an "all-vowel" substring. We advance j every time we see a consonant.

i is the current position, and k indicates the smallest window with all 5 vowels.

Note that we use a trick to move k one step forward, so the smallest window is actually [k - 1, j].

This is just to simplify the calculation, so that k - j == 1 when we find the first vowel substring.

So, for each position i, we have k - j valid substrings. The picture below demonstrate it for "xxaiioueiiaxx" test case:



int countVowelSubstrings(string w) {

int vow = 0, cnt = 0, m[123] = {};

string vowels("aeiou");

for (int i = 0, j = 0, k = 0; i < w.size(); ++i) {

if (vowels.find(w[i]) != string::npos) {

vow += ++m[w[i]] == 1;

for (; vow == 5; ++k)

vow -= --m[w[k]] == 0;

cnt += k - j;

}

else {

m['a'] = m['e'] = m['i'] = m['o'] = m['u'] = vow = 0;

j = k = i + 1;

}

}

return cnt;

}

**Java version**

class Solution {

    public int countVowelSubstrings(String word) {

        int j = 0, k = 0, vow = 0, cnt = 0;

        HashMap < Character, Integer > map = new HashMap < > ();

        map.put('a', 0);

        map.put('e', 0);

        map.put('i', 0);

        map.put('o', 0);

        map.put('u', 0);

        for (int i = 0; i < word.length(); ++i) {

            if (map.get(word.charAt(i)) != null) {

                map.put(word.charAt(i), map.get(word.charAt(i)) + 1);

                if ((int) map.get(word.charAt(i)) == 1) {

                    ++vow;

                }

                while (vow == 5) {

                    map.put(word.charAt(k), map.get(word.charAt(k)) - 1);

                    if ((int) map.get(word.charAt(k)) == 0) {

                        --vow;

                    }

                    k++;

                }

                cnt = cnt + (k - j);

            } else {

                map.forEach((k1, v) -> {

                    map.put(k1, 0);

                });

                vow = 0;

                j = k = i + 1;

            }

        }

        return cnt;

    }

}

**Refer to**

[L792.Number of Matching Subsequences (Ref.L208,L392,L1034,L2062)](note://WEB9e9521022575941e94e9a64861e1a74b)

[L992.Subarrays with K Different Integers](note://WEBed9f16531d534d22184e6335b24e4cff)

[L1513.Number of Substrings With Only 1s](note://WEB88cee6b65c48df54b2a36e150a94a79b)

[L1839.Longest Substring Of All Vowels in Order (Ref.L2062,L2401)](note://WEB6d3c6a99c1dc5283be2792571845e000)

[L2262.Total Appeal of A String (Ref.L828)](note://WEBb28ca12f4f1b056a97a22c14bef90a52)

[L2461.Maximum Sum of Distinct Subarrays With Length K (Ref.L2062,L2401)](note://WEB7bdebd65515808a4d25cc7c45804eb07)