<https://leetcode.com/problems/longest-substring-of-all-vowels-in-order/description/>

A string is considered **beautiful** if it satisfies the following conditions:

Each of the 5 English vowels ('a', 'e', 'i', 'o', 'u') must appear **at least once** in it.

The letters must be sorted in **alphabetical order** (i.e. all 'a's before 'e's, all 'e's before 'i's, etc.).

For example, strings "aeiou" and "aaaaaaeiiiioou" are considered **beautiful**, but "uaeio", "aeoiu", and "aaaeeeooo" are **not beautiful**.

Given a string word consisting of English vowels, return the **length of the longest beautiful substring** of word. If no such substring exists, return 0.

A **substring** is a contiguous sequence of characters in a string.

**Example 1:**

**Input:** word = "aeiaaioaaaaeiiiiouuuooaauuaeiu"

**Output:** 13

**Explanation:** The longest beautiful substring in word is "aaaaeiiiiouuu" of length 13.

**Example 2:**

**Input:** word = "aeeeiiiioooauuuaeiou"

**Output:** 5

**Explanation:** The longest beautiful substring in word is "aeiou" of length 5.

**Example 3:**

**Input:** word = "a"

**Output:** 0

**Explanation:** There is no beautiful substring, so return 0.

**Constraints:**

1 <= word.length <= 5 \* 10^5

word consists of characters 'a', 'e', 'i', 'o', and 'u'.

**Attempt 1: 2024-12-29**

**Solution 1: Greedy (10 min)**

**Style 1:**

class Solution {

    public int longestBeautifulSubstring(String word) {

        int maxLen = 0;

        int curLen = 1;

        int vowelCount = 1;

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

            if(word.charAt(i) == word.charAt(i - 1)) {

                curLen++;

            } else if(word.charAt(i) > word.charAt(i - 1)) {

                vowelCount++;

                curLen++;

            } else {

                curLen = 1;

                vowelCount = 1;

            }

            if(vowelCount == 5) {

                maxLen = Math.max(maxLen, curLen);

            }

        }

        return maxLen;

    }

}

Time Complexity: O(n)

Space Complexity: O(1)

**Refer to**

<https://leetcode.com/problems/longest-substring-of-all-vowels-in-order/solutions/1175189/best-c-solution/>

The basic idea is 'a' < 'e' < 'i' < 'o' < 'u'. So this problem is to find the length of the longest non-decreasing substring that has five different chars.

class Solution {

public:

int longestBeautifulSubstring(string word) {

const auto n = word.size();

int cnt = 1;

int len = 1;

int max\_len = 0;

for (int i = 1; i != n; ++i) {

if (word[i - 1] == word[i]) {

++len;

} else if (word[i - 1] < word[i]) {

++len;

++cnt;

} else {

cnt = 1;

len = 1;

}

if (cnt == 5) {

max\_len = max(max\_len, len);

}

}

return max\_len;

}

};

This is an awesome solution, but it wastes some time because a valid substring must start with 'a'.

Below is my improved code in Java. Runtime reduces to 11ms. Now beats 100%.

class Solution {

public int longestBeautifulSubstring(String word) {

char[] arr = word.toCharArray();

int maxLen = 0;

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

if (arr[i-1] == 'a') {

int currLen = 1, uniqueChars = 1;

while (i < word.length() && arr[i-1] <= arr[i]) {

uniqueChars += arr[i-1] < arr[i] ? 1 : 0;

currLen++;

i++;

}

if (uniqueChars == 5)

maxLen = Math.max(maxLen, currLen);

}

}

return maxLen;

}

}

**Style 2:**

class Solution {

    public int longestBeautifulSubstring(String word) {

        int maxLen = 0;

        int curLen = 0;

        int vowelCount = 0;

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

            if(i == 0 || word.charAt(i) >= word.charAt(i - 1)) {

                curLen++;

                if(i == 0 || word.charAt(i) > word.charAt(i - 1)) {

                    vowelCount++;

                }

                if(vowelCount == 5) {

                    maxLen = Math.max(maxLen, curLen);

                }

            } else {

                curLen = 1;

                vowelCount = 1;

            }

        }

        return maxLen;

    }

}

Time Complexity: O(n)

Space Complexity: O(1)

**Refer to chatGPT**

class Solution {

public int longestBeautifulSubstring(String word) {

int maxLen = 0; // Maximum length of valid substring

int currLen = 0; // Length of current valid substring

int vowelCount = 0; // Number of distinct vowels in the current substring

// Iterate through the characters in the string

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

// If it's the first character or the current character is not smaller than the previous one

if (i == 0 || word.charAt(i) >= word.charAt(i - 1)) {

// Increment the current substring length

currLen++;

// If we encounter a new vowel, increment the vowel count

if (i == 0 || word.charAt(i) != word.charAt(i - 1)) {

vowelCount++;

}

// Check if we have all 5 vowels in order

if (vowelCount == 5) {

maxLen = Math.max(maxLen, currLen);

}

} else {

// Reset for a new substring

currLen = 1;

vowelCount = 1;

}

}

return maxLen;

}

}

**Explanation:**

Key Idea:

To find the longest substring where all vowels ('a', 'e', 'i', 'o', 'u') appear in order and consecutively, iterate through the string and keep track of the length of valid substrings.

Tracking Vowels:

Use vowelCount to track how many distinct vowels are present in the current substring in the correct order.

Transition:

If the current character is greater than or equal to the previous character (lexicographically), extend the current substring.

If a new vowel is encountered (different from the previous character), increment vowelCount.

Reset:

If the order breaks (i.e., the current character is smaller than the previous one), reset currLen and vowelCount to start a new substring.

Max Length:

Update maxLen whenever all 5 vowels are found in a valid substring.

**Complexity:**

Time Complexity: O(n), where n is the length of the string, since we traverse the string once.

Space Complexity: O(1), as no additional data structures are used.

**Solution 2: Two Pointers + Harsh Table (10 min)**

class Solution {

    public int longestBeautifulSubstring(String word) {

        int maxLen = 0;

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

        int i = 0;

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

            if(j >= 1 && word.charAt(j) < word.charAt(j - 1)) {

                set.clear();

                i = j;

            }

            set.add(word.charAt(j));

            if(set.size() == 5) {

                maxLen = Math.max(maxLen, j - i + 1);

            }

        }

        return maxLen;

    }

}

Time Complexity: O(n)

Space Complexity: O(n)

**[Java/Python 3] Sliding window codes, w/ brief explanation and analysis.**

**Refer to**

<https://leetcode.com/problems/longest-substring-of-all-vowels-in-order/solutions/1175134/JavaPython-3-Sliding-window-codes-w-brief-explanation-and-analysis./>

Loop through input string and maintain a sliding window to fit in beautiful substrings;

Use a Set seen to check the number of distinct vowels in the sliding window;

Whenever adjacent letters is not in alpahbetic order, start a new window and new Set;

For each iteration of the loop, add the correponding char and check the size of the Set seen; If the size reaches 5, update the longest length;

Return the longest length after the termination of the loop.

public int longestBeautifulSubstring(String word) {

int longest = 0;

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

for (int lo = -1, hi = 0; hi < word.length(); ++hi) {

if (hi > 0 && word.charAt(hi - 1) > word.charAt(hi)) {

seen = new HashSet<>();

lo = hi - 1;

}

seen.add(word.charAt(hi));

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

longest = Math.max(longest, hi - lo);

}

}

return longest;

}

**Refer to**

[L2401.Longest Nice Subarray (Ref.L424,L2024)](note://WEB7538a2bbd5f8e4e2b3e803ba68408e0d)

[L2062.Count Vowel Substrings of a String](note://WEBffe1facb0ba61c1f4e6681e23ad781cb)