345. Reverse Vowels of a String

String

Problem Description

Two Pointers

they can be in both lowercase and uppercase. The goal is to reverse the order of only the vowels in the string without altering the 'o' have been reversed in their positions.

The problem presents a challenge to reverse the vowels in a given string s. Vowels include the characters 'a', 'e', 'i', 'o', and 'u', and

position of the other characters. For instance, given the string "hello", the function should return "holle" since the vowels 'e' and Intuition

Easy

swapped. The process continues until the two pointers meet or pass one another, which would mean all vowels have been considered. We arrive at this solution because it allows us to reverse the vowels in just one pass through the string, and we don't need additional data structures to track the vowels. Using the two-pointer technique is efficient because we only swap when

The intuition behind the solution is to use the two-pointer technique. One pointer (i) starts at the beginning of the string, while

the other pointer (j) starts at the end. The pointers move towards each other, and when they each find a vowel, those vowels are

necessary, meaning we ignore non-vowel characters and avoid unnecessary operations. To implement this, we first convert the string to a list because strings are immutable in Python, and we need to be able to swap the characters. We then iterate with our two pointers, checking for vowels and swapping them when both pointers point to

vowels. At the end, we convert the list back to a string and return it. Solution Approach

The solution is implemented using the two-pointer technique, along with basic list operations in Python. Here's the step-by-step

Convert the string to a list: Strings in Python are immutable, which means they cannot be changed after they are created. To

uppercase vowels ("aeiouAEIOU").

vowels: and decrement j if it's not a vowel.

approach:

efficiently swap elements, the string s is converted into a list of characters, cs = list(s). Initialize two pointers: We declare two pointers i and j. Pointer i is initialized to the start of the list (0), and pointer j is set to the end of the list (len(s) - 1).

- Create a set of vowels: To facilitate the quick lookup, we create a string vowels, which contains all the lowercase and
- Iterate and swap vowels: We enter a while loop that continues to iterate as long as i is less than j. Within this loop, we do the following:

Advance the i pointer until it points to a vowel by using a while loop: while i < j and cs[i] not in vowels: and

Once both pointers are at vowels, swap the vowels: cs[i], cs[j] = cs[j], cs[i]. Then, move both pointers closer to

- increment i if it's not a vowel. Similarly, move the j pointer backwards until it points to a vowel using a while loop: while i < j and cs[j] not in
- the center by incrementing i and decrementing j: i, j = i + 1, j 1. Continue until pointers meet: The pointer movements and swaps continue until i is no longer less than j, i.e., until they have

met or crossed each other, indicating that all vowels have been processed and potentially swapped.

Initial configuration: cs = ['l', 'e', 'e', 't', 'c', 'o', 'd', 'e'] i = 0, j = 7

• Now, i points to 'a', which is a vowel, we don't move it until we find a vowel for j to swap with.

- Convert list back to string and return: Once the loop is finished, the list cs contains the characters of the original string with the vowels reversed. We simply join the characters in cs back into a string: return "".join(cs) and return the resultant string.
- Illustration with an example: Let's consider the string s = "leetcode". We expect the function to return "leotcede" after reversing the vowels.
- After iterating and swapping: cs = ['l', 'e', 'o', 't', 'c', 'e', 'd', 'e'] i = 2, j = 5Finally, the list will look like this: cs = ['l', 'e', 'o', 't', 'c', 'e', 'd', 'e']And after joining it back into a string: Return value = "leotcede"

The algorithm has a time complexity of O(n), where n is the length of the string, as it involves a single pass through the string.

The space complexity is O(n) as well, due to the additional list used to hold the string characters for swapping.

'a' and 'i' get reversed.

Example Walkthrough

Let's apply the solution approach using a smaller example, string s = "Rain". The expected output would be "Rian" as the vowels

The set of vowels we'll use for lookup is "aeiouAEIOU".

Simultaneously:

Python

class Solution:

• j points to 'n', which is not a vowel, so j is decremented to 2. • Now, j points to 'i', which is a vowel, so we move on to swapping.

This walkthrough illustrates the step-by-step process as described in the solution approach. The desired vowels are reversed,

As i is no longer less than j, the loop ends. We convert the array back into a string: Return value is "Rian" after we perform "".join(cs).

and the string's non-vowel characters remain in their original positions.

Convert the string to a list to allow modification of characters.

Move both pointers inward to continue the process.

// Check if the pointers haven't crossed; swap the vowels if needed.

// Convert the character array back into a string and return it.

// After swapping, move both pointers to continue to the next characters.

Continue swapping until the two pointers cross.

Join the list back into a string and return it.

We now have vowels at both pointers, so we swap the values at i and j:

Initial configuration: cs = ['R', 'a', 'i', 'n'] i = 0, j = 3

• i points to 'R', which is not a vowel, so i is incremented to 1.

We start iterating with the following steps:

• We increment i and decrement j: i = 2, j = 1.

def reverse_vowels(self, s: str) -> str:

while left_pointer < right_pointer:</pre>

left_pointer += 1

left_pointer += 1

right_pointer -= 1

return "".join(char_list)

if (i < j) {

++i;

--j;

char temp = characters[i];

characters[j] = temp;

return String.valueOf(characters);

// Return the string with vowels reversed

const vowels = new Set(['a', 'e', 'i', 'o', 'u', 'A', 'E', 'I', '0', 'U']);

// Move the left pointer towards the right until it points to a vowel

// Move the right pointer towards the left until it points to a vowel

while (leftPointer < rightPointer && !vowels.has(characters[leftPointer])) {</pre>

while (leftPointer < rightPointer && !vowels.has(characters[rightPointer])) {</pre>

// Convert the string to an array to manipulate characters

// Process the characters until the two pointers meet

// Swap the vowels at the left and right pointers

// Join the characters back to a string and return the result

function reverseVowels(s: string): string {

const characters = s.split('');

// Initialize two pointers

let leftPointer = 0;

// Create a set of vowels for easy access

let rightPointer = characters.length - 1;

while (leftPointer < rightPointer) {</pre>

leftPointer++;

rightPointer--;

return characters.join('');

return str;

characters[i] = characters[j];

char_list = list(s)

Solution Implementation

Move the left pointer to the right until a vowel is found or pointers cross.

Move the right pointer to the left until a vowel is found or pointers cross.

while left_pointer < right_pointer and char_list[right_pointer] not in vowels:</pre>

while left_pointer < right_pointer and char_list[left_pointer] not in vowels:</pre>

• cs[i], cs[j] = cs[j], cs[i] translates to cs[1], cs[2] = cs[2], cs[1], resulting in the array ['R', 'i', 'a', 'n'].

Define a string containing all vowels in both lowercase and uppercase. vowels = "aeiouAEI0U" # Start pointers at the beginning and end of the string. left_pointer, right_pointer = 0, len(s) - 1

right_pointer -= 1 # Swap the vowels at left and right pointers if they haven't crossed. if left_pointer < right_pointer:</pre> char_list[left_pointer], char_list[right_pointer] = char_list[right_pointer], char_list[left_pointer]

```
Java
class Solution {
    public String reverseVowels(String s) {
       // Create an array to flag vowels. The ASCII value of the character will serve as the index.
       boolean[] isVowel = new boolean[128];
       // Populate the isVowel array with true for all vowel characters, both uppercase and lowercase.
        for (char c : "aeiouAEIOU".toCharArray()) {
            isVowel[c] = true;
       // Convert the input string to a character array for easy manipulation.
       char[] characters = s.toCharArray();
       // Initialize two pointers, one at the start (i) and one at the end (j) of the character array.
       int i = 0, j = characters.length - 1;
       // Use a while loop to traverse the character array from both ends until they meet or cross.
       while (i < j) {
           // Move the start pointer forward if the current character is not a vowel.
           while (i < j && !isVowel[characters[i]]) {</pre>
                ++i;
           // Move the end pointer backward if the current character is not a vowel.
           while (i < j && !isVowel[characters[j]]) {</pre>
```

};

TypeScript

```
C++
#include <string>
#include <algorithm>
class Solution {
public:
    // Function to reverse vowels in a given string
    std::string reverseVowels(std::string str) {
        // Initialize a simple hash array to mark vowels
        bool isVowel[128] = {false};
        // Mark each vowel as true in the hash array
        for (char c : "aeiouAEIOU") {
            isVowel[c] = true;
        // Initialize two pointers, one at the start and one at the end of the string
        int left = 0, right = str.size() - 1;
        // Use a two-pointer approach to find the vowels to be swapped
        while (left < right) {</pre>
            // Move the left pointer forward until a vowel is found
            while (left < right && !isVowel[str[left]]) {</pre>
                ++left;
            // Move the right pointer backward until a vowel is found
            while (left < right && !isVowel[str[right]]) {</pre>
                --right;
            // If both pointers still have not met or crossed, swap the vowels
            if (left < right) {</pre>
                std::swap(str[left++], str[right--]);
```

```
[characters[leftPointer], characters[rightPointer]] = [characters[rightPointer], characters[leftPointer]];
// Move the pointers closer towards the center
leftPointer++;
rightPointer--;
```

```
class Solution:
   def reverse_vowels(self, s: str) -> str:
       # Define a string containing all vowels in both lowercase and uppercase.
        vowels = "aeiouAEIOU"
       # Start pointers at the beginning and end of the string.
        left_pointer, right_pointer = 0, len(s) - 1
        # Convert the string to a list to allow modification of characters.
        char_list = list(s)
       # Continue swapping until the two pointers cross.
       while left_pointer < right_pointer:</pre>
            # Move the left pointer to the right until a vowel is found or pointers cross.
            while left_pointer < right_pointer and char_list[left_pointer] not in vowels:</pre>
                left_pointer += 1
            # Move the right pointer to the left until a vowel is found or pointers cross.
            while left_pointer < right_pointer and char_list[right_pointer] not in vowels:</pre>
                right_pointer -= 1
            # Swap the vowels at left and right pointers if they haven't crossed.
            if left_pointer < right_pointer:</pre>
                char_list[left_pointer], char_list[right_pointer] = char_list[right_pointer], char_list[left_pointer]
                # Move both pointers inward to continue the process.
                left_pointer += 1
                right_pointer -= 1
       # Join the list back into a string and return it.
        return "".join(char_list)
Time and Space Complexity
```

while loops inside it (for checking whether the characters at the pointers i and j are vowels or not) also contribute to the total time complexity.

Time Complexity

On average, each character in the string will be checked at most twice to see if it is a vowel (once for each pointer), which gives us O(2n) time complexity, but when simplified, it is denoted as O(n), where n is the length of the string. An additional consideration is the reversal operation for vowels which occurs once for each vowel in the string. However, since

The primary operation of the algorithm is the while loop that scans the string from both ends towards the center. The two nested

characters are vowels), it does not affect the overall time complexity which remains O(n).

this is a constant time operation (swapping characters) and is done at most n/2 times (assuming the worst case where all

Space Complexity

The space complexity is mainly due to converting the string to a list of characters, which takes O(n) space, where n is the length of the string. No additional data structures are used that grow with the size of the input, so no further space is required.

Thus, the space complexity of the code is O(n) because it needs to store all characters of the input string in a separate list to perform the swapping operation.