JavaScript Longest Substring Without Repeating Characters

Challenge

Given a string `s`, find the length of the longest substring without repeating characters.

1st Example

```
Input: s = 'abcabcbb'
Output: 3
Explanation: The answer is 'abc', with the length of 3.
```

2nd Example

```
Input: s = 'bbbbb'
Output: 1
Explanation: The answer is 'b', with the length of 1.
```

3rd Example

```
Input: s = 'pwwkew'
Output: 3
Explanation: The answer is 'wke', with the length of 3.
```

> **Important**
>
> The answer must be a substring, please note that `'pwke'` is a subsequence and not a substring.

Constraints

- `0 <= s.length <= 5 * 10⁴`
- `s` consists of English letters, digits, symbols and spaces.

Solution

```
const lengthOfLongestSubstring = (s) => {
    let set     = new Set(),
        left    = 0,
        maxSize = 0;

    if (s.length === 0) {
        return 0;
    }

    if (s.length === 1) {
        return 1;
    }
```

Solution continues on next page...

```
    for (let i = 0; i < s.length; i++) {
        while (set.has(s[i])) {
            set.delete(s[left]);

            left++;
        }

        set.add(s[i]);

        maxSize = Math.max(maxSize, i - left + 1);
    }

    return maxSize;
};
```

Explanation

I've defined a function called `lengthOfLongestSubstring` that calculates the length of the longest substring without repeating characters in a given string.

Inside the function, a set called `set` is initialized to store unique characters, and two variables, `left` and `maxSize`, are initialized to keep track of the left index of the current substring and the maximum length of the substring, respectively.

The function first checks if the length of the input string is `0` or `1`. If it is, it immediately returns `0` or `1` respectively since there can be no repeated characters in such cases.

Next, the function enters a loop that iterates through each character of the input string.

Inside the loop, there is another loop that runs as long as the set `set` already contains the current character `s[i]`. This inner loop is used to remove characters from the set until there are no repeating characters in the current substring.

In each iteration of the inner loop, it deletes the leftmost character of the current substring from the set using `set.delete(s[left])` and increments the `left` index to move to the next character.

Once there are no repeating characters in the current substring, it adds the current character `s[i]` to the set using `set.add(s[i])`.

After adding the character to the set, it calculates the length of the current substring by subtracting the left index from the current index and adding 1 (`i - left + 1`).

It then updates the `maxSize` variable with the maximum value between the current `maxSize` and the length of the current substring using `Math.max(maxSize, i - left + 1)`.

After iterating through all the characters of the input string, the function returns the `maxSize`, which represents the length of the longest substring without repeating characters.

In summary, the `lengthOfLongestSubstring` function finds the length of the longest substring without repeating characters in a given string by using a set to keep track of unique characters and two pointers to track the current substring.

Author: Trevor Morin