JavaScript Find All Anagrams in a String

Challenge

Given two strings `s` and `p`, return an array of all the start indices of `p`'s anagrams in `s`. You may return the answer in any order.

An Anagram is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

1st Example

```
Input: s = 'cbaebabacd', p = 'abc'
Output: [0,6]
Explanation: The substring with start index = 0 is
             'cba', which is an anagram of 'abc'.
             The substring with start index = 6 is
             'bac', which is an anagram of 'abc'.
```

2nd Example

```
Input: s = 'abab', p = 'ab'
Output: [0,1,2]
```

Example continues on next page...

```
Explanation: The substring with start index = 0 is
             'ab', which is an anagram of 'ab'.
             The substring with start index = 1 is
             'ba', which is an anagram of 'ab'.
             The substring with start index = 2 is
             'ab', which is an anagram of 'ab'.
```

Constraints

- `1 <= s.length, p.length <= 3 * 10⁴`
- `s` and `p` consist of lowercase English letters.

Solution

```
const findAnagrams = (s, p) => {
    const output      = [],
          neededChars = {};

    for (let char of p) {
        if (char in neededChars) {
            neededChars[char]++;
        } else neededChars[char] = 1;
    }

    let left  = 0,
        right = 0,
        count = p.length;
```

Solution continues on next page...

```
    while (right < s.length) {
        if (neededChars[s[right]] > 0) count--;

        neededChars[s[right]]--;
        right++;

        if (count === 0) output.push(left);

        if (right - left == p.length) {
            if (neededChars[s[left]] >= 0) count++;

            neededChars[s[left]]++;
            left++;
        }
    }

    return output;
};
```

Explanation

I've built a function called `findAnagrams` that takes in two parameters, `s` and `p`. The purpose of this function is to find all the anagrams of string `p` in string `s` and return an array of indices where the anagrams start in `s`.

This function begins by initializing an empty array called `output` and an empty object called `neededChars`. The `output` array will store the indices of the anagrams found, while the `neededChars` object will keep track of the characters needed to form an anagram.

Next, the function iterates through each character `char` in string `p`. For each character, it checks if `char` already exists as a key in `neededChars`. If it does, the value associated with that key is incremented by `1`. If `char` does not exist as a key, it is added to `neededChars` with a value of `1`.

After initializing some variables (`left`, `right`, and `count`) to control a sliding window approach, the function enters a while loop that continues until the `right` pointer reaches the end of string `s`.

Within the loop, it checks if the count of the character at `s[right]` in `neededChars` is greater than `0`. If it is, the `count` variable is decremented. The count of the character at `s[right]` in `neededChars` is also decremented. The `right` pointer is then incremented to move the sliding window to the right.

If the count variable becomes `0`, it means an anagram is found. In this case, the current value of `left` is added to the `output` array.

If the size of the sliding window (determined by `right - left`) becomes equal to the length of string `p`, it means we need to shrink the window from the left. In this case, the function checks if the count of the character at `s[left]` in `neededChars` is greater than or equal to `0`. If it is, the `count` variable is incremented. The count of the character at `s[left]` in `neededChars` is also incremented. The `left` pointer is then incremented to move the sliding window to the right.

Finally, this function returns the `output` array containing the indices where the anagrams start in `s`.

Author: Trevor Morin