Daily Coding Problem #320

Problem

This problem was asked by Amazon.

Given a string, find the length of the smallest window that contains every distinct character. Characters may appear more than once in the window.

For example, given "jiujitsu", you should return 5, corresponding to the final five letters.

Solution

This problem can be solved using a two pointer approach, where we keep track of `start` and end indices that bound the optimal window.

We will we iterate through the array, updating a dictionary that stores the count of each character. Once all the distinct characters have been seen, we can begin to narrow down our window.

Let us examine the procedure for a given end index, where we know that each character has previously appeared at least once. To begin, our start index will be zero, and the window ranging from `start` to end will be our best guess at a solution.

If the count of the character at our `start` index is greater than one, we know that character will reappear later on in the window. As a result, we can move this index forward. In fact,

we can keep incrementing this index, as long as doing so will not remove a distinct character. Finally, once we have shifted to the right as far as possible, we can take note of the current window size, `end - start`, and update our result if this window is smaller than the best result so far.

By following this procedure for each index in our string, we are guaranteed to find the smallest window in a single pass.

```
def window(string):
    string = list(string)

    char_seen = 0
    char_total = len(set(string))
    char_counts = defaultdict(int)

    start = 0
    result = float('inf')

    for end, char in enumerate(string):
        char_counts[char] += 1
        if char_counts[char] == 1:
            char_seen += 1

        if char_seen == char_total:
            while char_counts[string[start]] > 1:
                char_counts[string[start]] -= 1
                start += 1

            window_length = end - start + 1
            if window_length < result:
                result = window_length

    return result
```

Note that we can only increment the start counter `N` times in total, where `N` is the length of the string, so the inner while loop can be considered constant. Since the other operations are also constant for each character in the string, the time complexity of this algorithm is `O(N)`.

The space complexity will be `O(N)` as well, since there may be up to `N` keys we must store in the character count dictionary.