Longest Repeating Character Replacement

Difficulty	Medium
Category	Sliding Window
Question	https://leetcode.com/problems/longest-repeating-character-replacement/
Solution	https://youtu.be/gqXU1UyA8pk
Status	Done

Question

> You are given a string `s` and an integer `k`. You can choose any character of the string and change it to any other uppercase English character. You can perform this operation at most `k` times.
>
> Return *the length of the longest substring containing the same letter you can get after performing the above operations*.

Example

Example 1:

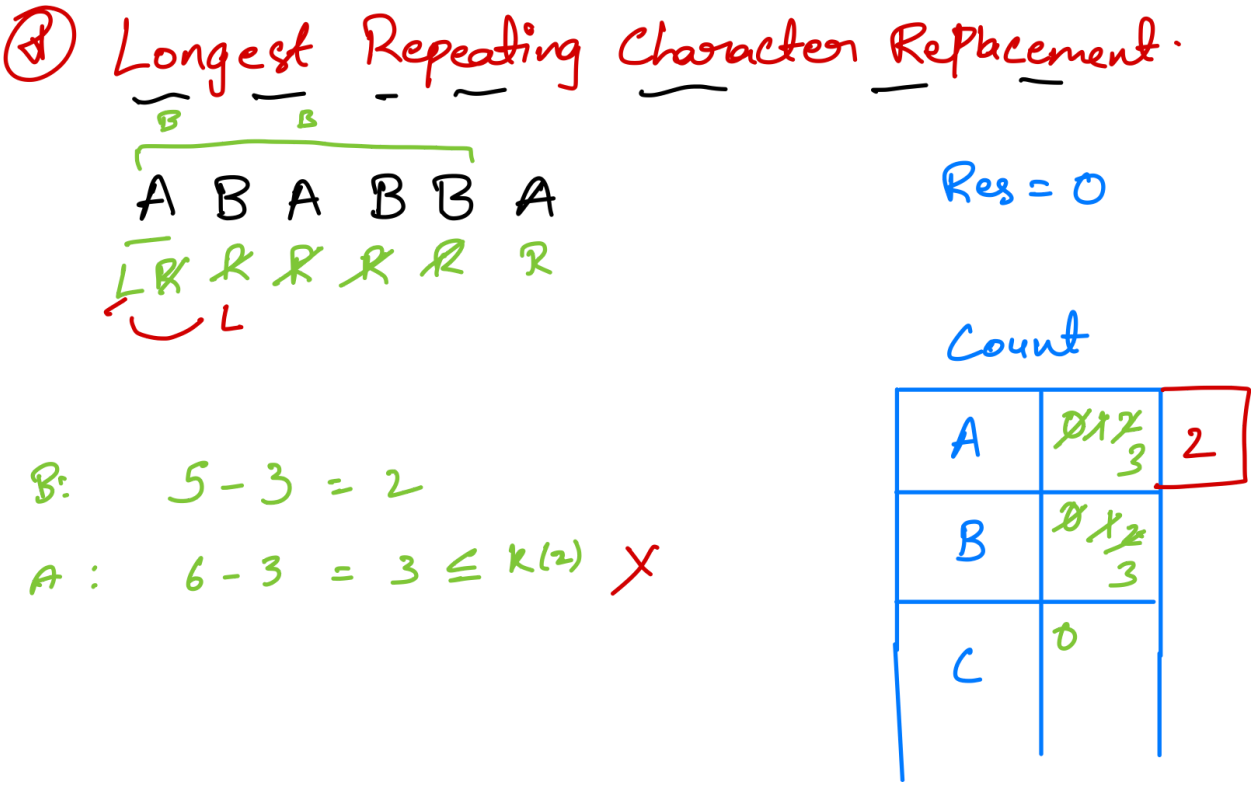
```
Input: s = "ABAB", k = 2
Output: 4
Explanation: Replace the two 'A's with two 'B's or vice versa.
```

Example 2:

```
Input: s = "AABABBA", k = 1
Output: 4
Explanation: Replace the one 'A' in the middle with 'B' and form "AABBBBA".
The substring "BBBB" has the longest repeating letters, which is 4.
There may exists other ways to achive this answer too.
```

Idea

Create a Dict, to maintain the count of each char in String(s). Check the difference between the sliding window and the Most frequent character and return the max_res



Solution

```
class Solution:
    def characterReplacement(self, s: str, k: int) -> int:
        # Create a dictionary to keep track of character counts.
        count = {}

        # Initialize the result variable.
        res = 0

        # Initialize left pointer (l) at the beginning of the string.
        l = 0

        # Iterate through the string using a sliding window (right pointer, r).
        for r in range(len(s)):
            # Update the count of the current character.
            count[s[r]] = 1 + count.get(s[r], 0)

            # Check if the window size minus the most frequent character count
            # exceeds the allowed replacements (k).
            while (r - l + 1) - max(count.values()) > k:
                # Move the left pointer (shrink the window) and update counts accordingly.
                count[s[l]] -= 1
                l += 1

            # Update the result with the maximum window size seen so far.
            res = max(res, r - l + 1)

        # Return the maximum window size, which represents the longest substring
        # with at most 'k' replacements.
        return res
```

Explanation

1. `count.get(s[r], 0)` is a dictionary method `get` that retrieves the value associated with the key `s[r]` in the `count` dictionary. If the key is not found (i.e., the character is not yet in the dictionary), it returns a default value of `0`.

The intuition behind this is to maintain a valid window by removing characters from the left side until you have at most `k` characters that are not the most frequent character.

This ensures that the window represents a substring where you can make at most `k` replacements to convert other characters into the most frequent character.

By continuously updating the left pointer and recalculating the window size and character counts, the algorithm identifies the longest substring that satisfies this condition, and the result is stored in the `res` variable. This way, you find the maximum length of a substring with at most `k` replacements.