Longest Repeating Character Replacement

Given a string that consists of only uppercase English letters, you can replace any letter in the string with another letter at most *k* times. Find the length of a longest substring containing all repeating letters you can get after performing the above operations.

Note:
Both the string's length and *k* will not exceed 10^4.

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.
```

Solution 1

The problem says that we can make at most k changes to the string (any character can be replaced with any other character). So, let's say there were no constraints like the k. Given a string convert it to a string with all same characters with minimal changes. The answer to this is

> length of the entire string - number of times of the maximum occurring character in the string

Given this, we can apply the at most k changes constraint and maintain a sliding window such that

> (length of substring - number of times of the maximum occurring character in the substring) <= k

```
class Solution {
public:
    int characterReplacement(string s, int k) {
        vector<int> counts(26, 0);
        int start = 0;
        int maxCharCount = 0;
        int n = s.length();
        int result = 0;
        for(int end = 0; end < n; end++){
            counts[s[end]-'A']++;
            if(maxCharCount < counts[s[end]-'A']){
                maxCharCount = counts[s[end]-'A'];
            }
            while(end-start-maxCharCount+1 > k){
                counts[s[start]-'A']--;
                start++;
                for(int i = 0; i < 26; i++){
                    if(maxCharCount < counts[i]){
                        maxCharCount = counts[i];
                    }
                }
            }
            result = max(result, end-start+1);
        }
        return result;
    }
};
```

written by harunrashidanver original link here

Solution 2

```
public int characterReplacement(String s, int k) {
    int len = s.length();
    int[] count = new int[26];
    int start = 0, maxCount = 0, maxLength = 0;
    for (int end = 0; end < len; end++) {
        maxCount = Math.max(maxCount, ++count[s.charAt(end) - 'A']);
        while (end - start + 1 - maxCount > k) {
            count[s.charAt(start) - 'A']--;
            start++;
        }
        maxLength = Math.max(maxLength, end - start + 1);
    }
    return maxLength;
}
```

There's no edge case for this question. The initial step is to extend the window to its limit, that is, the longest we can get to with maximum number of modifications. Until then the variable **start** will remain at 0.

Then as **end** increase, the whole substring from 0 to**end** will violate the rule, so we need to update **start** accordingly (slide the window). We move**start** to the right until the whole string satisfy the constraint again. Then each time we reach such situation, we update our max length.

written by Joshua924 original link here

Solution 3

Based on the Python solution by @dalwise. Use a sliding window `s[i:j]`, always add the new character, and remove the first window character if the extension isn't ok. So in each step, either extend the window by one or move it by one.

```
int characterReplacement(string s, int k) {
    int i = 0, j = 0, ctr[91] = {};
    while (j < s.size()) {
        ctr[s[j++]]++;
        if (j-i - *max_element(ctr+65, ctr+91) > k)
            ctr[s[i++]]--;
    }
    return j - i;
}
```

written by StefanPochmann original link here