

Longest Repeating Character Replacement

Use at most K changes to make the longest uniform window ■

● MEDIUM

■ Hook

You have a string of uppercase letters and you can replace at most K characters. Goal: make the longest possible substring where all characters are the same. This problem teaches a brilliant insight: instead of thinking 'what do I change?', think '**what do I keep?**'

■ The Key Insight — Count Replacements Needed

In a window of size len , if the most-frequent character appears $maxFreq$ times, then you need to change $len - maxFreq$ other characters to match it. The window is valid as long as $len - maxFreq \leq K$.

■ The Formula

$replacements_needed = window_size - max_frequency_in_window$ Valid window:
 $replacements_needed \leq K$ Equivalently: $window_size - maxFreq \leq K$

■ Why $maxFreq$ never decreases

We never shrink $maxFreq$ when the window shrinks. This might seem wrong, but it's safe: a smaller $maxFreq$ would only shrink the window further, and we already have the best answer from when $maxFreq$ was higher. We only care about growing the window.

■ Visual — String: AABABBA, $K=1$

```
String: "A A B A B B A"  K=1
      0 1 2 3 4 5 6

right=0: [A]      freq={A:1} maxFreq=1  len=1  1-1=0 ≤ 1 ✓
right=1: [AA]     freq={A:2} maxFreq=2  len=2  2-2=0 ≤ 1 ✓
right=2: [AAB]    freq={A:2,B:1} maxF=2  len=3  3-2=1 ≤ 1 ✓
right=3: [AABA]   freq={A:3,B:1} maxF=3  len=4  4-3=1 ≤ 1 ✓ ★
right=4: [AABAB]  freq={A:3,B:2} maxF=3  len=5  5-3=2 > 1 ✗ SHRINK
      remove A: left=1 [ABAB]  freq={A:2,B:2} len=4  4-3=1 ✓
right=5: [ABABB]  freq={A:2,B:3} maxF=3  len=5  5-3=2 > 1 ✗ SHRINK
      left=2 [BABB]  len=4  4-3=1 ✓
right=6: [BABBA]  freq={A:1,B:3} maxF=3... wait maxF=3? len=5 5-3=2 > 1 ✗
      SHRINK: left=3 [ABBA] len=4

★ ANSWER = 4 (e.g. 'AABA' replace B → 'AAAA')
```

■ Code Implementation (Java)

```
public class LongestRepeatingCharReplacement {

    public int characterReplacement(String s, int k) {
        int[] freq = new int[26]; // frequency of each char in window
        int left = 0, maxFreq = 0, maxLen = 0;

        for (int right = 0; right < s.length(); right++) {
            freq[s.charAt(right) - 'A']++;
            maxFreq = Math.max(maxFreq, freq[s.charAt(right) - 'A']);

            // If invalid window: (window_size - maxFreq) > k
            if ((right - left + 1) - maxFreq > k) {
                freq[s.charAt(left) - 'A']--; // shrink left
                left++;

                // Note: we don't update maxFreq here – intentional!
            }

            maxLen = Math.max(maxLen, right - left + 1);
        }

        return maxLen;
    }
}
```

■■ Time & Space Complexity

■ TIME COMPLEXITY

$O(n)$

■ SPACE COMPLEXITY

$O(26) = O(1)$

Single pass with two pointers. Frequency array of size 26 (fixed). Space is $O(1)$.

■ Common Mistakes

■ Trying to recalculate maxFreq after shrinking

Many solutions try to recompute maxFreq by scanning the freq array after each shrink — that makes it $O(26n)$. The trick is: you don't need to. If maxFreq doesn't grow, the window won't grow, and we only care about growth.

■ Practice Problems

#	Problem	Difficulty	Pattern
1	Longest Repeating Character Replacement	Medium	SW + Freq + maxFreq trick
2	Max Consecutive Ones III	Medium	SW (flip 0s)
3	Longest Subarray of 1s After Deleting One Element	Medium	SW
4	Minimum Number of Operations to Make Array Continuous	Hard	SW + Sorting
5	Minimum Swaps to Group All 1s Together	Medium	Fixed SW

■ Quick Revision

- ✓ Key formula: $\text{replacements_needed} = \text{window_size} - \text{maxFreq}$
- ✓ Valid window: $(\text{right} - \text{left} + 1) - \text{maxFreq} \leq K$
- ✓ maxFreq only grows — never decremented after shrink (intentional)
- ✓ freq[] array of size 26 tracks character frequencies
- ✓ When window is invalid, move left forward by 1
- ✓ Time: $O(n)$ | Space: $O(1)$

■ MENTAL MODEL — Remember This Forever

Don't think 'which characters do I change?' — instead think 'which character do I keep?' In a window, the dominant character stays, and at most K others get converted to match it. The window is valid as long as the 'minority characters' count doesn't exceed K .