LeetCode 424: Longest Repeating Character Replacement

■ Problem Understanding

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

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

■ Brute Force Approach

Idea: Try every possible substring and check how many replacements are needed to make all characters same. Track the maximum valid substring.

Time Complexity: $O(n^3)$
Space Complexity: O(1)

■ Brute Force Code

C++:
#include using namespace std; int characterReplacement(string s, int k) { int n = s.size(); int ans = 0; for (int i = 0; i < n; i++) { vector freq(26, 0); for (int j = i; j < n; j++) { freq[s[j] - 'A']++; int maxFreq = *max_element(freq.begin(), freq.end()); int len = j - i + 1; if (len - maxFreq <= k) ans = max(ans, len); } } return ans; } int main() { cout << characterReplacement("ABAB", 2); }

Java:
class Solution { public int characterReplacement(String s, int k) { int n = s.length(); int ans = 0; for (int i = 0; i < n; i++) { int[] freq = new int[26]; for (int j = i; j < n; j++) { freq[s.charAt(j) - 'A']++; int maxFreq = 0; for (int f : freq) maxFreq = Math.max(maxFreq, f); int len = j - i + 1; if (len - maxFreq <= k) ans = Math.max(ans, len); } } return ans; } }

Python:
def characterReplacement(s: str, k: int) -> int: n = len(s) ans = 0 for i in range(n): freq = [0] * 26 for j in range(i, n): freq[ord(s[j]) - ord('A')] += 1 max_freq = max(freq) if (j - i + 1) - max_freq <= k: ans = max(ans, j - i + 1) return ans print(characterReplacement("ABAB", 2))

■■ Optimal Approach — Sliding Window

Idea: Use a sliding window to expand while valid and shrink when invalid. We maintain character frequencies and keep track of the most frequent character inside the window.

Condition: (window_length - maxFreq) <= k
Time Complexity: O(n)
Space Complexity: O(1)

C++:
#include using namespace std; int characterReplacement(string s, int k) { vector freq(26, 0); int left = 0, maxFreq = 0, ans = 0; for (int right = 0; right < s.size(); right++) { freq[s[right] - 'A']++; maxFreq = max(maxFreq, freq[s[right] - 'A']); while ((right - left + 1) - maxFreq > k) { freq[s[left] - 'A']--; left++; } ans = max(ans, right - left + 1); } return ans; } int main() { cout << characterReplacement("ABAB", 2); }

Java:
class Solution { public int characterReplacement(String s, int k) { int[] freq = new int[26]; int left = 0, maxFreq = 0, ans = 0; for (int right = 0; right < s.length(); right++) { freq[s.charAt(right) - 'A']++; maxFreq = Math.max(maxFreq, freq[s.charAt(right) - 'A']); while ((right - left + 1) - maxFreq > k) { freq[s.charAt(left) - 'A']--; left++; } ans = Math.max(ans, right - left + 1); } return ans; } }

Python:
def characterReplacement(s: str, k: int) -> int: freq = [0] * 26 left = 0 max_freq = 0 ans = 0 for right in range(len(s)): freq[ord(s[right]) - ord('A')] += 1 max_freq = max(max_freq, freq[ord(s[right]) - ord('A')]) while (right - left + 1) - max_freq > k: freq[ord(s[left]) - ord('A')] -= 1 left += 1 ans = max(ans, right - left + 1) return ans print(characterReplacement("ABAB", 2))

■ Dry Run Example: s = 'AABABBA', k = 1

Step	left	right	window	freq(max)	condition	action	ans
1	0	0	A	A=1	1-1 ≤ 1 ■	expand	1
2	0	1	AA	A=2	2-2 ≤ 1 ■	expand	2
3	0	2	AAB	A=2	3-2=1 ≤1 ■	expand	3
4	0	3	AABA	A=3	4-3=1 ≤1 ■	expand	4
5	0	4	AABAB	A=3	5-3=2 ■	shrink	4
6	1	4	ABAB	A=2	4-2=2 ■	shrink	4
7	2	4	BAB	B=2	3-2=1 ■	expand	4
8	2	5	BABB	B=3	4-3=1 ■	expand	4
9	2	6	BABBA	B=3	5-3=2 ■	shrink	4	Final Answer = 4