Maximum Occuring Character GFG

Given a string, find the character that occurs the maximum number of times in the string. If multiple characters have the same maximum count, return the character with the lowest ASCII value.

Example:

Input: "testcase"

Output: 'e'

Explanation: In the input string, both 't' and 'e' occur twice. However, according to the problem statement, if multiple characters have the same maximum count, we should return the character with the lowest ASCII value. The ASCII value of 'e' is lower than that of 't', so the output is 'e'.

Approach 1: Function using unordered_map to find the character with maximum occurrence

- We use an **unordered_map** (hash map) to store the count of each character in the input string.
- We iterate through the string, incrementing the count of each character in the map.
- Then, we find the character with the highest count by iterating through the map. If two characters have the same count, we select the one with the lower ASCII value.
- **The time complexity of this approach is O(n), where n is the length of the input string.**
- **The space complexity is also O(n) since the hash map may store all unique characters in the worst case.**

Approach 2: Function using map to find the character with maximum occurrence

- Similar to the first approach, we use a **map** (ordered map) to store the count of each character in the input string.
- We iterate through the string, incrementing the count of each character in the map.
- Then, we find the character with the highest count by iterating through the map. If two characters have the same count, we select the one with the lower ASCII value.
- **The time complexity of this approach is O(n log n), where n is the length of the input string.**

- **The space complexity is O(n) since the map may store all unique characters in the worst case.**

Approach 3: Optimized function using an array to find the character with maximum occurrence

- In this approach, we use an array of size 26 (assuming lowercase letters) to store the count of each character.
- We iterate through the string and increment the count of the corresponding character in the array.
- Then, we find the character with the highest count by iterating through the string again. If two characters have the same count, we select the one with the lower ASCII value.
- **The time complexity of this approach is O(n), where n is the length of the input string.**
- **The space complexity is O(1) since the array size is fixed and does not depend on the input string length.**

Which approach to use:

Among the three approaches, the optimized array-based approach is recommended for most scenarios due to its better time and space complexity. It provides a balance between efficiency and simplicity.