<https://leetcode.com/problems/reorganize-string/description/>

Given a string s, rearrange the characters of s so that any two adjacent characters are not the same.

Return *any possible rearrangement of* s *or return* "" *if not possible*.

**Example 1:**

Input: s = "aab"

Output: "aba"

**Example 2:**

Input: s = "aaab"

Output: ""

**Constraints:**

1 <= s.length <= 500

s consists of lowercase English letters.

**Attempt 1: 2023-02-06**

**Solution 1:  Hash Table (30 min)**

**Style 1: Use map**

class Solution {

public String reorganizeString(String s) {

Map<Character, Integer> map = new HashMap<Character, Integer>();

for(int i = 0; i < s.length(); i++) {

char c = s.charAt(i);

map.put(c, map.getOrDefault(c, 0) + 1);

}

int maxFreq = 0;

char maxChar = '\*';

for(Map.Entry<Character, Integer> entry : map.entrySet()) {

if(entry.getValue() > maxFreq) {

maxFreq = entry.getValue();

maxChar = entry.getKey();

}

}

// If maximum characater frequency over half, not possible

// and "+1" is important especially for odd length string

if(maxFreq > (s.length() + 1) / 2) {

return "";

}

char[] result = new char[s.length()];

int index = 0;

while(maxFreq > 0) {

result[index] = maxChar;

index += 2;

maxFreq--;

}

// Remove the maxChar from map since all used to fill in 'result' already

map.remove(maxChar);

/\*\*

Consider this example: "aaabbbcdd", we will construct the string in this way:

a \_ a \_ a \_ \_ \_ \_ // fill in "a" at position 0, 2, 4

a b a \_ a \_ b \_ b // fill in "b" at position 6, 8, 1

a b a c a \_ b \_ b // fill in "c" at position 3

a b a c a d b d b // fill in "d" at position 5, 7

\*/

// Fill all empty buckets with remain chars as key in map

for(char c : map.keySet()) {

while(map.get(c) > 0) {

// When reach the end come back to fill in again all empty buckets

if(index >= s.length()) {

index = 1;

}

result[index] = c;

index += 2;

map.put(c, map.get(c) - 1);

}

}

return new String(result);

}

}

**Style 2: Use int[26]**

class Solution {

public String reorganizeString(String s) {

int n = s.length();

int[] freq = new int[26];

for(char c : s.toCharArray()) {

freq[c - 'a']++;

}

int maxFreq = 0;

int maxFreqIndex = 0;

for(int i = 0; i < 26; i++) {

if(freq[i] > maxFreq) {

maxFreq = freq[i];

maxFreqIndex = i;

}

}

// If maximum characater frequency over half, not possible

if(maxFreq > (n + 1) / 2) {

return "";

}

/\*\*

Consider this example: "aaabbbcdd", we will construct the string in this way:

a \_ a \_ a \_ \_ \_ \_ // fill in "a" at position 0, 2, 4

a b a \_ a \_ b \_ b // fill in "b" at position 6, 8, 1

a b a c a \_ b \_ b // fill in "c" at position 3

a b a c a d b d b // fill in "d" at position 5, 7

\*/

char[] result = new char[n];

int index = 0;

while(freq[maxFreqIndex] > 0) {

result[index] = (char)(maxFreqIndex + 'a');

index += 2;

freq[maxFreqIndex]--;

}

for(int i = 0; i < 26; i++) {

while(freq[i] > 0) {

// Very smart part to continue assign from start positions

if(index >= n) {

index = 1;

}

result[index] = (char)(i + 'a');

index += 2;

freq[i]--;

}

}

return new String(result);

}

}