/\*\*

 \* @param {string} s

 \* @return {string}

 \*/

var reorganizeString = function(s) {

    // 1. 统计字符频率

    const freqMap = new Map();

    for (const char of s) {

        freqMap.set(char, (freqMap.get(char) || 0) + 1);

    }

    // 2. 创建最大堆（按频率排序）

    const maxHeap = new MaxHeap();

    for (const [char, count] of freqMap) {

        maxHeap.insert({ char, count });

    }

    // 3. 重构字符串

    let result = [];

    let prev = null; // 记录上一个使用的字符

    while (maxHeap.size() > 0) {

        // 取出当前频率最高的字符

        const first = maxHeap.extractMax();

        // 如果前一个字符和当前字符相同，需要取出次高频率的字符

        if (prev && prev.char === first.char) {

            if (maxHeap.size() === 0) {

                return ""; // 无法满足条件

            }

            const second = maxHeap.extractMax();

            result.push(second.char);

            prev = second;

            if (second.count > 1) {

                maxHeap.insert({ char: second.char, count: second.count - 1 });

            }

            maxHeap.insert(first); // 把第一个字符重新放回堆中

        } else {

            result.push(first.char);

            prev = first;

            if (first.count > 1) {

                maxHeap.insert({ char: first.char, count: first.count - 1 });

            }

        }

    }

    return result.join("");

};

// 最大堆实现（按频率排序）

class MaxHeap {

    constructor() {

        this.heap = [];

    }

    size() {

        return this.heap.length;

    }

    insert(val) {

        this.heap.push(val);

        this.bubbleUp(this.heap.length - 1);

    }

    extractMax() {

        const max = this.heap[0];

        const end = this.heap.pop();

        if (this.heap.length > 0) {

            this.heap[0] = end;

            this.bubbleDown(0);

        }

        return max;

    }

    bubbleUp(index) {

        while (index > 0) {

            const parentIndex = Math.floor((index - 1) / 2);

            if (this.heap[parentIndex].count >= this.heap[index].count) break;

            [this.heap[parentIndex], this.heap[index]] = [this.heap[index], this.heap[parentIndex]];

            index = parentIndex;

        }

    }

    bubbleDown(index) {

        const length = this.heap.length;

        while (true) {

            const leftChildIndex = 2 \* index + 1;

            const rightChildIndex = 2 \* index + 2;

            let largestIndex = index;

            if (leftChildIndex < length && this.heap[leftChildIndex].count > this.heap[largestIndex].count) {

                largestIndex = leftChildIndex;

            }

            if (rightChildIndex < length && this.heap[rightChildIndex].count > this.heap[largestIndex].count) {

                largestIndex = rightChildIndex;

            }

            if (largestIndex === index) break;

            [this.heap[index], this.heap[largestIndex]] = [this.heap[largestIndex], this.heap[index]];

            index = largestIndex;

        }

    }

}

