【第二十周】专项面试题解析-下半节

395. 至少有 K 个重复字符的最长子串

这是典型的分治思想。出现的频次小于k的得到的答案一定不能横跨字符串。所以将原字符串的字符进行频度统计，并按照频度小于k的字符进行分割，递归进行求解，每个字段中符合题意的

```
var longestSubstring = function(s, k) {
    const n = s.length;

    // 对原始字符串s的切割过程
    return dfs(s, k);
}

// 对字符串进行tokenize，每个片段重复计算是否满足要求
const dfs = (s, k) => {
    if(!s) return 0;

    const cnt = new Array(26).fill(0);
    // 统计当前片段中字符出现频度
    for (const ch of s) {
        cnt[ch.charCodeAt() - 'a'.charCodeAt()]++;
    }


   for(let i=0;i<26;i++){
       if(cnt[i] && cnt[i]<k){
        const tokens=s.split(String.fromCharCode(i+'a'.charCodeAt()))
           let ret=0;

        for(const token of tokens){
            const len=dfs(token,k)
            ret=Math.max(len,ret)
        }

        return ret;
       }
   }

    return s.length;
};
```

8. 字符串转换整数 (atoi)

借助 parseInt(string, radix)，parseInt() 会尽可能把字符串转换成数字

```
/**
```

```
 * @param {string} s
 * @return {number}
 */
// 一个字符串转换成一个整数，判断是否越界。
// 如果整数超过边界，就输出 边界值
// 就是考整数，最大的整数是2147483647
var myAtoi = function (str) {
    const number = parseInt(str, 10);
    const Max = Math.pow(2, 31) - 1;
    const Min = Math.pow(-2, 31);

    // 无法转换的情况返回 0
    if (isNaN(number)) {
        return 0;
    }
    // 转换结果超出范围的情况
    if (number < Min || number > Max) {
        return number < 0 ? Min : Max;
    }
    return number;
};
```

190. 颠倒二进制位

1.扫描0-32位，看看每一位是什么数字

2.如果当前数字的第一位是1，那么就把结果数字的第32位置为1

```
/**
 * @param {number} n - a positive integer
 * @return {number} - a positive integer
 */
var reverseBits = function(n) {
    let ret = 0;
    for (let i = 0; i < 32 && n > 0; ++i) {
        ret |= (n & 1) << (31 - i);
        n >>>= 1;
    }
    return ret >>> 0;
};
```

380. O(1) 时间插入、删除和获取随机元素

数组存值，易随机。哈希表存值 → 索引，易查询

1.插入：数组push值，哈希表值 → 数组长度 - 1
2.删除：哈希表值 → 索引，数组 索引 与 末位 交换，更新哈希表末位值 → 索引

（在哈希表中删除元素的流程：1.在哈希表中查找要删除元素的索引。2.将要删除元素与最后一个元素交换。3.删除最后一个元素。4.更新哈希表中的对应关系。）

3.数组pop，哈希表delete
4.随机：[0, 1) 伪随机数 * 数组长度取整 = 随机索引

```
 /**
 * Initialize your data structure here.
 */
var RandomizedSet = function() {
    this.h = {},this.a = [];
    // 可以参考林位财同学set做法
    // this.m = new Map();     this.data = [];

};

/**
 * Inserts a value to the set. Returns true if the set did not already contain
the specified element.
 * @param {number} val
 * @return {boolean}
 */
RandomizedSet.prototype.insert = function(val) {
    return this.h[val] === undefined && (this.a.push(val),this.h[val] =
this.a.length - 1,true);
};

/**
 * Removes a value from the set. Returns true if the set contained the specified
element.
 * @param {number} val
 * @return {boolean}
 */
RandomizedSet.prototype.remove = function(val) {
    return this.h[val] !== undefined && (
        [this.a[this.h[val]],this.a[this.a.length - 1]] = [this.a[this.a.length
- 1],this.a[this.h[val]]],
        this.h[this.a[this.h[val]]] = this.h[val],
        this.a.pop(),delete(this.h[val]),true
    )
};
    // 可以参考林位财同学set做法
    // this.m = new Map();     this.data = [];
// RandomizedSet.prototype.remove = function (val) {
//     if (!this.m.has(val)) return false;
//     let n = this.m.get(val);
//     let len = this.data.length - 1;
//     [this.data[n], this.data[len]] = [this.data[len], this.data[n]];
//     this.m.set(this.data[n], n);
//     this.data.pop();
//     this.m.delete(val);
//     return true;
// };


/**
 * Get a random element from the set.
```

```
 * @return {number}
 */
RandomizedSet.prototype.getRandom = function() {
    return this.a[Math.random() * this.a.length | 0];
};

/**
 * Your RandomizedSet object will be instantiated and called as such:
 * var obj = new RandomizedSet()
 * var param_1 = obj.insert(val)
 * var param_2 = obj.remove(val)
 * var param_3 = obj.getRandom()
 */
```

402. 移掉 K 位数字

这个题就是删除尽可能多的K个数字，让剩下的数字是最小值。

1.映射到单调栈就是将原序列的数字，从前往后扫描，依次压入栈里面，单调栈里面最多可以出栈k个元素，单调栈里面剩下的元素就是剩余的数字；

2.这个就是利用单调栈的原理，让尽可能小的数字移动到前面

```
var removeKdigits = function(num, k) {
    let n = num.length;
    if (n <= k || n === 1) return '0';

    const handleStr = (str) => { // 对前导0进行处理例如：“00200”
        let i = 0;
        while(str[i] == 0) i++;
        if (i == str.length) return '0'
        return str.slice(i)
    }
    let stack = [];
    let count = 0;

    for (let i = 0; i < n; i++) {
        while (stack.length && stack[stack.length - 1] > num[i]) { // 栈顶的值小于当前值，栈顶出栈
            stack.pop();
            count++;
            if (count === k) { // 当count === k" 直接返回
                return handleStr(stack.join('') + num.slice(i)) // 将栈里的元素和剩余未入栈的元素拼接后进行处理
            }
        }
        stack.push(num[i])
    }
    // num 为正序例如'12345678'情况，count < k从尾部直接截取
    if (count < k) return handleStr(stack.join('').slice(0, count - k));
};
```

1081. 不同字符的最小子序列

遍历字符串 s：若当前字符在栈中存在，不需要执行任何操作，直接继续遍历下一个字符，即如果遍历到当前栈中已经有的字符，可以舍弃当前遍历到的字符(因为要去除字符串中重复的字母，使得每个字母只出现一次)。

```
/**
 * @param {string} s
 * @return {string}
 */
var smallestSubsequence = function (s) {
    let arr = [];
    for (let i = 0; i <= s.length - 1; i++) {
        let str = s[i]
        if (arr.includes(str)) continue;
        while (arr.length > 0 && arr[arr.length - 1] > str &&
s.indexOf(arr[arr.length - 1], i) > i) {
            arr.pop()
        }
        arr.push(str)
    }
    return arr.join("")
};
```

1499. 满足不等式的最大值

因为 xj 是大于 xi 的，那么把公式变换一下： yi + yj + |xi - xj| => xj + yj + yi - xi
那么对于每一个 j 来说，只需要找到最大的 yi - xi 的值，来更新当前存储的最大值即可
使用队列：

1. 存储对于当前的 j 来说，满足条件 |xi - xj| <= k 的 points
2. 保持单调递减，保证队列头部就是队列中最大的 yi-xi

```
/**
 * @param {number[][]} points
 * @param {number} k
 * @return {number}
 */

var findMaxValueOfEquation = function(points, k) {
  let len = points.length,
      max = -Infinity,
      queue = [];

  for (let j = 0; j < len; j++) {
    let [xj, yj] = points[j];
```

```
    // 把队列头部不满足条件 |xi - xj| <= k 的元素 shift 掉
    while (queue.length > 0 && xj - queue[0][0] > k) queue.shift();
    // 更新最大值
    if (queue.length > 0) {
      max = Math.max(queue[0][1] - queue[0][0] + xj + yj, max);
    }
    // 在把当前的 points[j] push 加入到队尾之前，把队列尾部比 points[j] 的 yj-xj
    // 小的元素 pop 掉，保证队列单调递减
    while (queue.length > 0 && (queue[queue.length - 1][1] - queue[queue.length
- 1][0]) < (yj - xj)) queue.pop();
    queue.push( points[j] );
  }

  return max;
};
```