【第十二课】5月--月度测试刷题课

【彩蛋题目：OJ上面172题】

763. 划分字母区间

我们发现a区间包含b区间c区间，b区间和C区间是相交的。看后面，d区间和e区间相交；这个就是上节课合并区间的变种，要的起点和终点。这个要合并之后的大小。

```
  /**
 * @param {string} s
 * @return {number[]}
 */
var partitionLabels = function(s) {
    const last = new Array(26);
    const length = s.length;
    const codePointA = 'a'.codePointAt(0);
    for(let i = 0; i < length; i++){
        last[s.codePointAt(i) - codePointA] = i;
    }
    const partition = [];
    let start = 0, end = 0;
    for(let i = 0; i< length; i ++){
        end = Math.max(end,last[s.codePointAt(i) - codePointA]);
        if(i == end){
            partition.push(end - start + 1);
            start = end + 1;
        }
    }
    return partition;
};
```

面试题 16.16. 部分排序

思路：这个题就是找逆序对。第一个指针从前往后指，规则是升序，遇到一个逆序对，就把它给记录下来，例如一开始出现的11 和 7 。第二个指针从后往前前，规则是降序。

```
  /**
 * @param {number[]} array
 * @return {number[]}
 */
//  找逆序对
var subSort = function(array) {
    let r = -1,l = -1;
    // 从左往右找最右边的区间值
    let max = Number.MIN_SAFE_INTEGER;
    for(let i = 0; i < array.length;i++){
        if(array[i] >= max){
            max = array[i];
        }else{
            r = i;
        }
```

```
    }
    // 从右往左找 找最左边的区间值
    let min = Number.MAX_SAFE_INTEGER;
    for(let i = array.length -1; i >= 0;i--){
        if(array[i] <= min){
            min = array[i]
        }else{
            l = i;
        }
    }
    return [l,r];

};
```

687. 最长同值路径

路径长度 = 节点数 -1。看 第二个例子一条是4-4-4和5-5-5。我们递归+一个深搜很容易就解决掉；

递归函数是求一个子树可以向父节点提供的路径长度。

对于当前节点，左子树能提供的长度为 left，如果当前节点值等于左子节点的值，则左链的长度等于 left+1，否则为0。

对于当前节点，右子树能提供的长度为right，如果当前节点值等于右子节点的值，则右链的长度等于 right+1，否则为0。

当前子树对父节点提供的最大长度为左右链中较大的一个。当前子树的左右链之和，去和全局最大值比较，试图更新。

```
/**
 * Definition for a binary tree node.
 * function TreeNode(val, left, right) {
 *     this.val = (val===undefined ? 0 : val)
 *     this.left = (left===undefined ? null : left)
 *     this.right = (right===undefined ? null : right)
 * }
 */
/**
 * @param {TreeNode} root
 * @return {number}
 */
// 递归
var longestUnivaluePath = function(root) {
    let res = 0

    const dfs = (root) => {
        if (root == null) {
            return 0
        }
        const left = dfs(root.left)
        const right = dfs(root.right)

        let leftPath = 0, rightPath = 0

        if (root.left && root.left.val == root.val) {
            leftPath = left + 1
        }
```

```
        if (root.right && root.right.val == root.val) {
            rightPath = right + 1
        }
        res = Math.max(res, leftPath + rightPath)

        return Math.max(rightPath, leftPath)
    }

    dfs(root)
    return res
}
```

179. 最大数

思路： 要想组成最大的整数，一种直观的想法是把数值大的数放在高位。于是我们可以比较输入数组的每个元素的最高位，最高位相同的时候比较次高位，以此类推，完成排序，然后把它们拼接起来。

```
 /**
 * @param {number[]} nums
 * @return {string}
 */
var largestNumber = function(nums) {
    nums.sort((x,y) =>{
        let sx = 10,sy = 10;
        while(sx <= x){
                sx *= 10;
        }
        while(sy <= y){
            sy *= 10;
        }
        return '' + (sx * y + x) - ('' + (sy * x + y));
    })
    if(nums[0] === 0){
        return '0';
    }
    return nums.join('');
};
```

347. 前 K 个高频元素

思路： 用快排做这个，这题堆的时候已经讲过的：面试题17.14前k个最小数；十分相似就不讲了；还是定义分区思想，将数组分成已排序和待排序，看已排序的小于k,证明数不够，从后面拿过来；同理，看已排序的大于k,证明排好的数已经多了，从前面拿过来。

```
 /**
 * @param {number[]} nums
 * @param {number} k
 * @return {number[]}
 */
var topKFrequent = function (nums, k) {
    // 统计出现次数
    const map = {};
    for (const n of nums) {
        n in map || (map[n] = 0);
        map[n]++;
```

```
    }

    const list = Object.entries(map);
    quickSelect(list, k, 0, list.length - 1, function (item, pivot) {
        return item[1] >= pivot[1];
    });

    return list.slice(0, k).map(el => el[0]);
};

/**
 * 把 arr[r] 当成是 pivot
 * 把大于等于 pivot 的数字放到左边
 * 把小于 pivot 的数字放到右边
 * @param {number[]} arr
 * @param {number} l
 * @param {number} r
 */
function partition(arr, l, r, comparator) {
    if (typeof comparator != 'function') {
        comparator = function (num, pivot) {
            return num >= pivot;
        };
    }

    let i = l;
    for (let j = l; j < r; j++) {
        if (comparator(arr[j], arr[r])) {
            [arr[i], arr[j]] = [arr[j], arr[i]];
            i++;
        }
    }
    // 将 pivot 换到分界点
    [arr[i], arr[r]] = [arr[r], arr[i]];
    // 返回 pivot 的下标
    return i;
}

/**
 * 寻找第 k 大元素
 * 如果 pivot 的下标刚好是 k - 1，那我们就找到了
 * 如果下标大于 k - 1，那就在 [left, pivotIndex - 1] 这段找第 k 大元素
 * 如果下标小于 k - 1，那就对 [pivotIndex + 1, right] 这段找第 k - pivotIndex + left
- 1 大元素
 * @param {number[]} list
 * @param {number} left
 * @param {number} right
 * @param {number} k
 * @param {function} comparator
 */
function quickSelect(list, k, left = 0, right = list.length - 1, comparator) {
    if (left >= right) return list[left];
    const pivotIndex = partition(list, left, right, comparator);

    if (pivotIndex - left === k - 1) return list[pivotIndex];
    else if (pivotIndex - left > k - 1)
        return quickSelect(list, k, left, pivotIndex - 1, comparator);
    else
```

```
        return quickSelect(
            list,
            k - pivotIndex + left - 1,
            pivotIndex + 1,
            right,
            comparator,
        );
}
```

1647. 字符频次唯一的最小删除次数

统计出现的次数并降序，得到数组。从数组第二个开始循环，比较与前一个值比较大小，如果大于等于前一个值且不为0，则增加一次"操作次数"，最终操作次数则为最少删除次数。

```
 /**
 * @param {string} s
 * @return {number}
 */
var minDeletions = function(s) {
    let charArr = new Array(26).fill(0);
    let arr = s.split('');
    arr.forEach((i) => charArr[i.charCodeAt() - 97]++);
    let resArr = charArr.filter((i) => i > 0).sort((a,b) => b - a);
    console.log(resArr)
    let count = 0;
    for(let i = 1; i < resArr.length;i++){
        while(resArr[i] >= resArr[i - 1] && resArr[i] > 0){
            resArr[i]--;
            count++;
        }
    }
    return count;
};
```

283. 移动零

定义两个指针，快指针、慢指针。快指针用于遍历指向非0。如果非0，将其值给到慢指针指向的元素，慢指针右移。若是0则不作操作。当快指针走到头，所有不是0的元素都被赋值到数组的前面，从慢指针的位置开始，后面的数都设置为0。

```
 /**
 * @param {number[]} nums
 * @return {void} Do not return anything, modify nums in-place instead.
 */
var moveZeroes = function(nums) {
    let index = 0;
    for(let i = 0; i < nums.length; i++){
        if(nums[i] != 0){
            nums[index] = nums[i];
            index++;
        }
    }
    for(let i = index; i < nums.length; i++){
        nums[i] = 0;
```

```
    }
    return nums;
};
```

面试题 10.01. 合并排序的数组

两个数组倒着合并，每次取两者中的较大值把它放进去，合并完了之后正好就是这个长度。

```
/**
 * @param {number[]} A
 * @param {number} m
 * @param {number[]} B
 * @param {number} n
 * @return {void} Do not return anything, modify A in-place instead.
 */
var merge = function(A, m, B, n) {
    // 定义两个指针
    let pa = m - 1, pb = n - 1;
    // 定义一个总长度，合并到链表上的长度
    let tail = m + n - 1;
    // 定义合并后的数组
    var cur;
    // 只要两个有一个大于0.证明还能合并
    while (pa >= 0 || pb >= 0) {
        if (pa === -1) {
            cur = B[pb--];
        } else if (pa === -1) {//pa合并完了，合并pb
            cur = A[pb--];
        } else if (pb === -1) {//pb合并完了，合并pa
            cur = A[pa--];
        } else if (A[pa] > B[pb]) {//如果两个数组都没有合并完，看哪个数组大；如果a数组比b
数组大，合并pa
            cur = A[pa--];
        } else {//最后一组就合并pb数组
            cur = B[pb--];
        }
        //将cur存到a上面倒着去合并，这样所有的值都合并到A上面了
        A[tail--] = cur;
    }
};
```

561. 数组拆分 I

思路：发现怎么拆，让最小值总和才是最大呢， 把数组做升序排列，然后每两个拆成一对，既然都是最小值了，那我们数组中最大的元素就用不到，如何 让最小值总和才是最大呢，以第一个例子来讲，我可以用次大的元素和最大的元素相加，那么他俩之和就是最大值；除了他俩之外，同理倒着推，然后最大2+次大1；次小值之和最大，这个时候我们发现就是我们排序后，最小值之和最大就是下标为偶数元素总和；

我们看第二个例子，组成的结合就是: (6,6) ， (2,5) ,(1,2);=> 1,2,2,5,6,6 下标为偶数就是 1+2+6 = 9；

```
 /**
 * @param {number[]} nums
 * @return {number}
 */
var arrayPairSum = function(nums) {
    nums.sort((a, b) => a - b);
    let ans = 0;
    for (let i = 0; i < nums.length; i += 2) {
        ans += nums[i];
    }
    return ans;
};
```

1353. 最多可以参加的会议数目

首先把所有的会议按照结束时间排序，我们优先参加早结束的会议。接着由于一天只能参加一个会议，所以使用一个set 记录我们使用过的天。最后参加每一个会议时，优先使用比较早的天来参加。

```
   /**
 * @param {number[][]} events
 * @return {number}
 */

var maxEvents = function(events) {
  let set=new Set()
  events.sort((a, b) => a[1] - b[1]);
  let flag = true;
   for(let i = 1; i < events.length; i++) {
       if(events[i][1] == events[i - 1][1]) {
           flag = false;
           break;
       }
   }
   if(flag == true) {
       return events.length;
   }
  for(let event of events){
      for(let i=event[0];i<=event[1];i++){
          if(!set.has(i)){
              set.add(i)
              break;
          }
      }
  }
  return set.size
};
```