1、为了不断优化推荐效果，今日头条每天要存储和处理海量数据。假设有这样一种场景：我们对用户按照它们的注册时间先后来标号，对于一类文章，每个用户都有不同的喜好值，我们会想知道某一段时间内注册的用户（标号相连的一批用户）中，有多少用户对这类文章喜好值为k。因为一些特殊的原因，不会出现一个查询的用户区间完全覆盖另一个查询的用户区间(不存在L1<=L2<=R2<=R1)。

参考答案1

` let chaxunzuarr = []; let yonghushu = readline(), xihaoduarr = readline().split(' '), chaxunzushu = readline(); for(let i = 0;i { if(arr[item] == undefined){ arritem; } arr[item].push(index); }); for(let j = 0;j{ if(e>=start && e<=end){ geshu++; } }) print(geshu); } } `

参考答案2

这个题的主要问题就是使用顺序扫描用户的时候会花费比较长的时间，从而导致超时

为了节省顺序查找的时间，我们可以利用二分查找的思路来实现： 1.使用ArrrayList将喜好值相同的用户id存储起来（此时这个ArrayList的用户id是递增的） 2.使用hashmap将该喜好值为k和对应的用户id的list进行存储 3.每次查找时找到对应k的ArrayList进行二分查找即可

参考答案3

`js //给前面的回答添加了点注释，方便阅读 let chaxunzuarr = []; //第一行输入，用户个数 let yonghushu = readline(), //第二行输入，用户对应喜好，转化为数组 xihaoduarr = readline().split(' '), //第三行输入，查询组数 chaxunzushu = readline(); //循环所有查询组，4行开始的所有行 for(let i = 0;i { if(arr[item] == undefined){ arritem; } arr[item].push(index); });

//遍历查询组 for(let j = 0;j{ if(e>=start && e<=end){ geshu++; } }) print(geshu); } } `

2、作为一个手串艺人，有金主向你订购了一条包含n个杂色串珠的手串——每个串珠要么无色，要么涂了若干种颜色。为了使手串的色彩看起来不那么单调，金主要求，手串上的任意一种颜色（不包含无色），在任意连续的m个串珠里至多出现一次（注意这里手串是一个环形）。手串上的颜色一共有c种。现在按顺时针序告诉你n个串珠的手串上，每个串珠用所包含的颜色分别有哪些。请你判断该手串上有多少种颜色不符合要求。即询问有多少种颜色在任意连续m个串珠中出现了至少两次。

参考答案1

` // 初始化串珠总个数，连续的串珠个数，颜色种类数，所有串珠的颜色信息数组, 同一颜色的串珠数组, 不合格的颜色个数 let ballNums, linkNums, colorNums, ballColor = , count = 0; [ballNums, linkNums, colorNums] = readline().split(' ').map(item => Number(item));

// 依次保存每个串珠所用颜色信息 // 数组的每个元素都是一个数组，元素数组的第一位代表颜色个数，后续代表所用颜色 for(let i = 0; i < ballNums; ++i) { ballColor[i] = readline().split(' ').map(item => Number(item)) }

// 将同一颜色出现的串珠序号进行收集 ballColor.forEach((item, index) => { // 若该串珠所用颜色种类大于0 // console.log(item, 'item'); if(item[0] > 0) { let colorArr = item.slice(1) // 下面的item代表不同的颜色种类 colorArr.forEach(color => { // 如果之前已经保存过使用某颜色的串珠序号，则直接将其添加到数组中去 if(sameColorBall[color]) { sameColorBall[color].push(index + 1) } else { sameColorBallcolor } }) }

}) sameColorBall.forEach(item => { // sameColorBall[0]代表使用‘0’这种颜色的所有串珠的序号数组,这里的序号是按顺序排列的 for(let i = 0; i < item.length - 1; ++i) { if(itemi+1 < linkNums) { ++count; break; } if(ballNums - itemitem.length - 1 < linkNums) { ++count; break; } } })

console.log(count); `

参考答案2

存储每个颜色所在的珠子位置，然后将位置排序并遍历，若两个位置的间距小于m则此颜色不合格。速度非常快，仅用时6ms

参考答案3

/判断手串珠子颜色不合格/ var line1 = readline().split(' '); var n = line1[0], m = line1[1], c = line1[2]; var obj = {}; var arr = []; var cnt = 0; for(var i=0;i<n;i++){ arr[i] = readline().split(' '); for(var j=1;j<=arri;j++){ if(objarr[i]){ objarr[i].push(i+1); }else{ objarr[i] = [i+1]; } } } var temp = Object.values(obj); for(var i=0;i<temp.length;i++){ for(var j=0;j<temp[i].length-1;j++){ if((tempi - tempi) < m){ cnt++; break; } } if(n-tempi+tempi < m){ cnt++; break; } } print(cnt);

3、以下函数使用二分查找搜索一个增序的数组，当有多个元素值与目标元素相等时，返回最后一个元素的下标，目标元素不存在时返回-1。请指出程序代码中错误或不符最佳实践的地方（问题不止一处，请尽量找出所有你认为有问题的地方）

int BinarySearchMax(const std::vector& data, int target)

{

int left = 0;

int right = data.size();

while (left < right) {

int mid = (left + right) / 2;  
  
 if (data[mid] <= target)  
  
 left = mid + 1;  
  
 else  
  
 right = mid - 1;

}

if (data[right] == target)

return right;

return -1;

}

参考答案1

1、当data大小为零时，明显会出错，应该先判断一下。

2、 right 应该等于 data.size()-1，否则当目标出现在data最右边或比data中所有的数大时，会访问越界。

修改后的代码如下

int BinarySearchMax(const vector<int>& data, int target) { if (data.size() == 0) return -1; int left = 0; int right = data.size() - 1; while (left < right) { int mid = (left + right) / 2; if (data[mid] <= target) left = mid + 1; else right = mid - 1; } if (data[right] == target) return right; return -1; }

参考答案2

` int BinarySearchMax(const std::vector& data, int target)

{

int left = 0;  
  
int right = data.size() - 1; // 最後一個index  
  
while (left < right) {  
  
 int mid = (left + right) / 2;  
 if (data[mid] == target) // mid的判斷應在這, 符合的話就可以結束運算  
 return mid;  
  
 if (data[mid] < target) // '=' 的話就已經從上方判斷結束，拿掉'='  
 left = mid + 1;  
  
 else  
  
 right = mid - 1;  
  
}   
  
return -1;

} `

4、设计一个TODO List，页面结构如下图所示，

使用HTML与CSS完成界面开发 实现添加功能：输入框中可输入任意字符，按回车后将输入字符串添加到下方列表的最后，并清空输入框 实现删除功能：点击列表项后面的“X”号，可以删除该项 实现模糊匹配：在输入框中输入字符后，将当前输入字符串与已添加的列表项进行模糊匹配，将匹配到的结果显示在输入框下方。如匹配不到任何列表项，列表显示空

注：以上代码实现需要能在浏览器中正常显示与执行。

图片: https://uploader.shimo.im/f/hjP3y96vClNo6W33.png!thumbnail?download=1&token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJjbHQiOiJleHBvcnQiLCJ1c2VySWQiOjAsImV4cCI6MTYyNjIwNzQ3N30.rM96djt14xk2wKhFKJj3nNivajFh2S8\_F6HAJ636rHA&accessToken=eyJhbGciOiJIUzI1NiIsImtpZCI6ImRlZmF1bHQiLCJ0eXAiOiJKV1QifQ.eyJhdWQiOiJhY2Nlc3NfcmVzb3VyY2UiLCJleHAiOjE2NDAwNTY2NjAsImciOiJ2R0NVWEZ1cWpjbzVQV01zIiwiaWF0IjoxNjQwMDU2MzYwLCJ1c2VySWQiOjE2NTQyNzAzfQ.GbBRhLY6WiMQZzip95hJFc4n6CDW842o0EpYxpLz\_hU

参考答案

`

<meta charset="utf-8">  
 <title>TODOList</title>  
 <style type="text/css">  
 body{  
 margin: 0;  
 background-color: #f5f5f5;  
 }  
 h1{  
 margin: 30px 0 0 0;  
 color: #ff5550;  
 text-align: center;  
 font-size: 60px;  
 }  
 #back{  
 width: 300px;  
 margin: 0 auto;  
 border: 1px solid #333;  
 box-shadow: 0px 0px 3px #999;  
 background-color: #fff;  
 }  
 #back input{  
 width: 100%;  
 box-sizing: border-box;  
 line-height: 30px;  
 border: none;  
 border-bottom: 1px solid #000;  
 padding: 5px 15px;  
 font-size: 18px;  
 }  
 #list\_back .single{  
 position: relative;  
 border-bottom: 1px solid #000;  
 }  
 #list\_back .single p{  
 width: 100%;  
 height: 30px;  
 margin: 0;  
 line-height: 30px;  
 padding: 5px 15px;  
 }  
 #list\_back .single span{  
 position: absolute;  
 right: 0;  
 top: 0;  
 width: 30px;  
 text-align: center;  
 line-height: 40px;  
 font-size: 18px;  
 color: #000;  
 cursor: pointer;  
 }  
 </style>

todos

<input id="addInput" type="text" name="">  
 <div id="list\_back">  
  
 </div>

`