> **L题解题报告：分块yyds**
>
> 题意：多次查询数列的区间的众数

朴素做法：针对每个区间用数组下标存个数，遍历遍区间，个数最大的那个元素的下标即为最大众数。

时间复杂度O（n2），必炸。

怎么办呢？

假如我们把整个数列分成若干个小块，

记录每个小块里众数是那个数字及其个数，

记录第i个小块到第j个小块里的众数及其个数

```
void build_f(){//f[i][j]:i~j的众数，它一定是f[i][j-1]的众数或者j-1分块里的某个数
    for(int i=1;i<=bl[n];++i){
        memset(num,0,sizeof num);
        int max_num=-1;
        int max_value=-1;
        for(int j=(i-1)*block+1;j<=n;++j){
            ++num[arr_deal[j]];
            if(num[arr_deal[j]]>max_value||(num[arr_deal[j]]==max_value&&arr_deal[j]
<max_num)){
                max_num=arr_deal[j];
                max_value=num[arr_deal[j]];
            }
            if(bl[j+1]!=bl[j]) f[i][bl[j]]=max_num;
        }
    }
}
```

后面查询的时候，

如果正巧区间的两端均位于我们分块的端点处，

那么根据我们直接记录的结果。

就可以直接得出答案了。

事实情况可能并没有那么巧合，

比如说左端点在s1块，右端点在s5块，

中间有s2,s3,s4三个块。

起码我们知道中间三个块的众数是多少。

那么答案要么就是这个众数，要么就是s1块和s5块中零散的那些数字中的一个。

这时候该怎么判断哪个是众数？

总不能还是得遍历一遍吧，然后比较这些数出现的次数。

那和不分块有啥区别。

为了解决这个问题，我们需要引入额外的一个数组。

`cnt[i][j],表示第j个点在前i个块中的数量`

有了这个数组之后，我们计算散块中的数字在区间的数量的时候，

就不用再遍历中间完整的那些块了，而是借助这个数组直接计算出来。

这个数组怎么得到呢？

在输入数据的时候，同时统计每个数组在各个块出现的次数，

计算cnt[][]这个数组时就会方便很多。

如果是存在temp[i][j]的话，表示第j个点在第i个块中的数量，

那么根据公式：cnt[i][j]=cnt[i-1][j]+temp[i][j]，就可以得到该数组。

```
void build_cnt(){
    for(int i=1;i<=bl[n];++i){
        for(int j=1;j<=len;++j){
            cnt[i][j]=cnt[i-1][j]+temp_cnt[i][j];
            //printf("cnt[%d][%d]:%d\n",i,j,cnt[i][j]);
        }
    }
}
```

时间复杂度O（sqrt(n)*n)

下面来总结下解题思路：

我们把数列分成很多小块，每个单独小块的众数及其数量我们是知道的。

查询区间众数时，先找左端点和右端点所在分块；

如果在同一分块或者相邻分块，直接暴力遍历找众数，时间复杂度O(sqrt(n));

如果中间有完整小块，那么区间众数一定是该小块的众数或者散块中的数；

枚举散块中数，借助cnt[][]数组省去完整块的遍历，只遍历散块，找到区间内个数最多的那个数。

时间复杂度O(sqrt(n));

询问m次，时间复杂度O(m*sqrt(n))。