P4168 [Violet] 蒲公英

题目大意

共 40000 个int范围内数字，一共 50000 次询问，$[l,r]$范围内的众数，如果出现次数相同则优先输出更小的数，强制在线。

题目分析

首先，$O(n^2)$ 爆搜肯定会TLE，因此思考更高效的算法：考虑对这 40000 个数进行根号分块。

我们需要思考，哪些数有机会成为 $[l,r]$ 内的众数，这里给出一个Claim：对于中间的整块出现的数，只有这一整块的众数有机会成为整个区间的众数。其他需要另外考虑的则是在两端零散的段中出现过的数字，由分块的性质显然这些候选数字不超过 $2\sqrt{n}+1$个。

当我们选出这些候选数字后，可以很方便地通过前缀和在 $O(\sqrt{n})$内找出出现次数最多的那个数字。

接下来说明Claim成立：考虑某个整块内的数，它未在两端零散段中出现，且不是这些整块的众数。由这个性质，它在整块内出现次数一定小于等于众数（等于时，它的编号大于那个众数），又未在零散段中出现，因此它不可能成为 $[l,r]$ 范围的众数。

最后给出预处理的方式：我们需要处理两个数据，一个是指定数字出现次数的前缀和，一个是指定区间内的众数（最小单位为整块）。考虑到如果需要处理出任意下标范围的出现数字次数的前缀和，我们需要填 $O(n^2)$ 个数，这本身便会让复杂度过大，因此考虑对块做前缀和，这部分处理的复杂度为 $O(n\sqrt{n})$。对于指定区间内众数，考虑到我们只需要用到区间范围为整块倍数大小的众数，因此这里只需要以整块为单位处理。通过fix左端点为整块的开头，移动右端点，可以在 $O(n\sqrt{n})$ 时间内处理出整块区间的众数。

此外有一些细节需要注意：数字的范围为 $1e9$ 以内，直接开cnt数组显然会爆空间，因此需要对数字进行离散化。预处理复杂度 $O(n\sqrt{n})$，询问复杂度 $O(\sqrt{n})$，总体复杂度为 $O(n\sqrt{n})$。下面给出代码示例：

离散化与分块：

```
int n, m, sq;
int a[N];

struct Block{
    int st, ed;
}block[sq_N];
```

```
int belong[N];
inline void make_block(){
    //分块操作
    sq = sqrt(n);
    for(int i=1;i<=sq;i++){
        block[i].st = n / sq * (i - 1) + 1;
        block[i].ed = n / sq * i;
    }
    block[sq].ed = n;
    for(int i=1;i<=sq;i++)
        for(int j=block[i].st;j<=block[i].ed;j++)
            belong[j] = i;
}

int value[N], tot;
unordered_map<int, int> ranker;

inline void discrete(){
    //离散化
    for(int i=1;i<=n;i++) value[i] = a[i];
    sort(value + 1, value + 1 + n);
    tot = unique(value + 1, value + 1 + n) - value - 1;
    for(int i=1;i<=tot;i++) ranker[value[i]] = i;
}
inline void input(){
    n = read(), m = read();
    for(int i=1;i<=n;i++) a[i] = read();
    make_block();
    discrete();
    for(int i=1;i<=n;i++) a[i] = ranker[a[i]];
}
```

预处理：

```
int times[sq_N][N];
int common[sq_N][sq_N];

inline void process(){
    //计算单块内每个数字出现次数
    for(int i=1;i<=sq;i++)
        for(int j=block[i].st;j<=block[i].ed;j++)
            times[i][a[j]]++;
    //进行前缀和
    for(int i=1;i<=sq;i++)
        for(int j=1;j<=tot;j++)
            times[i][j] += times[i - 1][j];
```

```
    unordered_map<int, int> cnt;
    for(int i=1;i<=sq;i++){//fix左端点
        int bigid = tot, bigcnt = 0;
        for(int j=1;j<=tot;j++) cnt[j] = 0;
        for(int j=i;j<=sq;j++){//移动右端点
            for(int k=block[j].st;k<=block[j].ed;k++){
                cnt[a[k]]++;
                if(cnt[a[k]] > bigcnt or (cnt[a[k]] >= bigcnt and a[k] < bigid)){
                    bigid = a[k];
                    bigcnt = cnt[a[k]];
                }
            }
            common[i][j] = bigid;
        }
    }
}
```

询问操作:

```
unordered_set<int> candidate;//候选数字
unordered_map<int, int> final_cnt//每个数字出现的次数
inline int query(int l, int r){
    for(auto ele : candidate){//每轮结束后清除上一次操作痕迹
        final_cnt[ele] = 0;
    }
    candidate.clear();

    if(abs(belong[l] - belong[r]) <= 1){//如果不存在整块，直接算
        for(int i=l;i<=r;i++){
            candidate.insert(a[i]);
            final_cnt[a[i]]++;
        }
    }
    else{
        candidate.insert(common[belong[l] + 1][belong[r] - 1]);//取整块众数为候选数字
        for(int i=l;i<=block[belong[l]].ed;i++){//以及两端零散的数字
            candidate.insert(a[i]);
            final_cnt[a[i]]++;
        }
        for(int i=block[belong[r]].st;i<=r;i++){
            candidate.insert(a[i]);
            final_cnt[a[i]]++;
        }
        for(auto ele : candidate){
            final_cnt[ele] += (times[belong[r] - 1][ele] - times[belong[l]][ele]);
        }
    }
```

```
    int bigid = tot, bigcnt = 0;
        for(auto ele : candidate){
            if(final_cnt[ele] > bigcnt or (final_cnt[ele] >= bigcnt and ele < bigid)){
                bigid = ele;
                bigcnt = final_cnt[ele];
            }
        }
    return bigid;
}
```