2020暑假牛客多校第五场H Interval

给定一个序列A，长度为N，定义函数$F(l,r)=A_l\&A_{l+1}\&\ldots\&A_r$，集合$S(l,r)=\{F(a,b)|min(l,r)\leqslant a\leqslant b\leqslant max(l,r)\}$，也就是对$[l,r]$的所有子区间求$F$的值并去重，题目给出序列$A$，以及$Q$个询问，每个询问给出$l$，$r$，求$S(l,r)$的元素个数。

该题目强制在线，$L=(L'\oplus lastans)\%N+1$，$R=(R'\oplus lastans)\%N+1$。

考虑对于1到N的位置建立普通线段树，维护每个位置出现的不同数字个数。对于序列前x个元素，当查询的区间右界$R=x$的时候，若$F(y,x)=k$，那么对于任意$L\leqslant y$都满足$k\in S(L,x)$，因此对于每一个数字F的值，只要维护它最靠右的出现位置即可，而且对于每一个F，只能出现在一个位置，不可以重复计数。对于数字的去重，可以通过unordered_map来实现。

根据上述分析，查询$S(L,R)$的时候，需要在$[1,R]$上建立的线段树中查询$[L,R]$的区间，因此需要对每一个位置为区间右界构造线段树，为了更高效，采用主席树来实现。显然，当$R=1$的时候，整个线段树有且只有一个位置有1的权值，而对于任意$R'=R+1$，新的区间的所有F值必然包含原来的区间，因此对$[1,R]$建树的时候所求得的$S(1,R)$需要进行记录，而对于$S(1,R')$，除了包含$S(1,R)$的所有元素以外，还额外包含了$A_{R'}$以及$S(1,R)$中的所有元素与$A_{R'}$按位与的结果，将这些新的元素加入$S(1,R)$去重后即可得到$S(1,R')$。在去重的时候需要始终维护所有数字只保留出现位置最靠后的一个。在主席树创建一个新树的时候，添加的新元素与已经存在的元素发生重复，需要在新树上进行修改，在该元素之前出现的位置进行update（-1）的操作，在该元素更新后的位置进行update（1）的操作，也就是修改其最后出现的位置。

建树完成之后，每次查询只要在$root[R]$的树上对区间$[L,R]$进行查询即可。

```
#include<bits/stdc++.h>
#include<unordered_map>
#define mid (l+r)>>1
using namespace std;
const int maxn = 30000005;
int ls[maxn], rs[maxn], val[maxn], root[100005];//左右儿子，权值，主席树的不同根
unordered_map<int, int>mp, la, tmp;//当前树去重得到的元素集合，上一棵树的元素集合，临时辅助集合
int tot;//中结点个数
int newnode(int rt, int v)//动态开点
{
    val[++tot] = val[rt] + v;//直接在开点的时候修改权值
    ls[tot] = ls[rt];
    rs[tot] = rs[rt];
    return tot;
}
void update(int& now, int la, int pos, int v, int l, int r)//更新，la代表上一棵树的同位置根节点
{
    if (l > pos || r < pos)
        return;
    now = newnode(la, v);//动态开点
    if (l == r)return;
    int m = mid;
    update(ls[now], ls[la], pos, v, l, m);
    update(rs[now], rs[la], pos, v, m + 1, r);
}
int query(int now, int L, int R, int l, int r)//普通二叉树区间查询
```

```
{
    if (l > R || r < L)return 0;
    if (l >= L && r <= R)return val[now];
    int m = mid;
    return query(ls[now], L, R, l, m) + query(rs[now], L, R, m + 1, r);
}
int main()
{
    int n;
    cin >> n;
    for (int i = 1; i <= n; i++)
    {
        int x;
        scanf("%d", &x);
        root[i] = root[i - 1];//复制新根
        mp[x] = i;//插入新的数字x，位置为i
        tmp.clear();
        for (auto it : mp)
        {
            tmp[it.first & x] = max(tmp[it.first & x], it.second);//计算出所有新增F的值，存在tmp中
        }
        for (auto it : tmp)
        {
            if (la[it.first] == it.second)continue;//如果某元素已经存在过了，而且位置没有发生改变，就不进行更新
            update(root[i], root[i], la[it.first], -1, 1, n);//删去原来的位置
            la[it.first] = it.second;//在存放历史树信息的集合中更新数据
            update(root[i], root[i], la[it.first], 1, 1, n);//插入在新的位置
        }
        mp.swap(tmp);//更新mp集合
    }
    int q;
    int lastans = 0;
    cin >> q;
    while (q--)
    {
        int l, r;
        scanf("%d%d", &l, &r);
        l = (l ^ lastans) % n + 1;//强制在线
        r = (r ^ lastans) % n + 1;
        if (l > r)swap(l, r);
        lastans = query(root[r], l, r, 1, n);//查询
        printf("%d\n", lastans);
    }
    return 0;
}
```