猫树

算法简介

一种 ST 表的进阶数据结构。支持 $O(n\log n)$ 预处理，$O(n)$ 修改，$O(1)$ 查询。

算法实现

考虑分治处理每个区间询问 $[ql,qr]$ 的过程。设当前分治区间为 $[L,R]$，左区间为 $[L,M]$，右区间为 $[M+1,R]$。

如果 $[ql,qr]\subseteq [L,M]$ 或 $[ql,qr]\subseteq [M+1,R]$，则递归处理。否则一定有询问左端点位于左区间，右端点位于右区间。

于是可以预处理出左区间后缀答案以及右区间前缀答案，这样一个询问只会被拆分成两个区间，然后一次合并即可得到结果。

考虑建立线段树模拟分治过程同时每个结点预处理左右区间的前后缀答案，显然时间复杂度为 $O(n\log n)$。

不难发现 $[ql,qr]$ 的最终需要查询的结点为 ql 所在叶子结点和 qr 所在叶子结点在线段树上的 LCA。

通过观察，不难发现，对于线段树编号 10100 和 10001 的结点，他们的 LCA 为 10，这也是他们线段树编号的 LCP。

通过异或编号同时预处理 $\log2(n)$ 可以快速找到 LCA 对应深度然后直接查询答案，时间复杂度为 $O(1)$。

上述结论只对线段树上相同深度的结点有效，最后为了保证所有叶子结点在线段树的同一深度，需要将序列长度补成 2 的幂次。

于是注意开两倍空间，另外算法本身的空间复杂度为 $O(n\log n)$。

最后关于修改，考虑暴力修改该点在每层的影响，时间复杂度 $O(\frac n2+\frac n4+\frac n8+\cdots)\sim O(n)$。

修改也可以考虑树状数组维护前缀和，不过这样貌似不如直接用线段树维护。

算法模板

区间最大子串和

SPOJ1043

```
const int MAXN=5e4+5,MAXD=20;
```

```
namespace Tree{
    int n,a[MAXN<<1],lg2[MAXN<<1],s1[MAXD][MAXN<<1],s2[MAXD][MAXN<<1];
    void build(int L,int R,int d){
        if(L==R)return;
        int M=L+R>>1,sum,ans;
        s1[d][M]=s2[d][M]=a[M];
        sum=a[M],ans=max(a[M],0);
        for(int i=M-1;i>=L;i--){
            sum+=a[i],ans+=a[i];
            s1[d][i]=max(s1[d][i+1],sum);
            s2[d][i]=max(s2[d][i+1],ans);
            ans=max(ans,0);
        }
        s1[d][M+1]=s2[d][M+1]=a[M+1];
        sum=a[M+1],ans=max(a[M+1],0);
        for(int i=M+2;i<=R;i++){
            sum+=a[i],ans+=a[i];
            s1[d][i]=max(s1[d][i-1],sum);
            s2[d][i]=max(s2[d][i-1],ans);
            ans=max(ans,0);
        }
        build(L,M,d+1);
        build(M+1,R,d+1);
    }
    void init(int _n){
        n=1;
        while(n<_n)n<<=1;
        _rep(i,2,n)lg2[i]=lg2[i>>1]+1;
        build(1,n,0);
    }
    int query(int L,int R){
        if(L==R)return a[L];
        int d=lg2[n]-lg2[(L-1)^(R-1)]-1;
        return max({s2[d][L],s2[d][R],s1[d][L]+s1[d][R]});
    }
}
int main(){
    int n=read_int();
    _rep(i,1,n)
    Tree::a[i]=read_int();
    Tree::init(n);
    int q=read_int();
    while(q--){
        int l=read_int(),r=read_int();
        enter(Tree::query(l,r));
    }
    return 0;
}
```