吉司机线段树

算法简介

一种支持区间最值修改及各种查询的线段树。

算法模板

模板一

HDU5306

题意

给定一个序列，支持下述三种操作：

1. 对 $l\le i\le r$，$a_i=\min(a_i,v)$
2. 询问区间 $[l,r]$ 的最大值
3. 询问区间 $[l,r]$ 的和

题解

考虑建立线段树，维护区间最大值，区间最大值个数，区间次大值，区间和。

对修改操作，如果v不小于区间最大值，则该操作没有意义，直接返回，否则先进行普通区间修改操作。

当普通区间修改操作遇到当前区间被包含于修改区间时，如果 v 的大于区间次大值，则直接修改区间最大值，同时打上懒标记。

如果 v 不大于区间次大值，则暴力修改子树。发现修改过程本质是 dfs，而 dfs 时间复杂度等于遍历结点个数。

定义一个结点所代表区间的不同数值所构成的集合为该结点的数集，则暴力修改至少会合并该结点的最大值和次大值，于是该结点数集大小减小。

普通区间修改操作至多会使该结点的数集增加一个值，但普通区间修改的结点只有 $O(\log n)$ 个。

初始时所有结点的数集大小和为 $O(n\log n)$，于是修改操作的总复杂度为 $O((n+m)\log n)$。

对查询操作，和普通区间查询操作完全相同。

```
const int MAXN=1e6+5;
int a[MAXN];
int
lef[MAXN<<2],rig[MAXN<<2],maxv[MAXN<<2],maxc[MAXN<<2],secv[MAXN<<2],lazy[MA
XN<<2];
```

```
LL sum[MAXN<<2];
void push_up(int k){
    sum[k]=sum[k<<1]+sum[k<<1|1];
    if(maxv[k<<1]==maxv[k<<1|1]){
        maxv[k]=maxv[k<<1];
        maxc[k]=maxc[k<<1]+maxc[k<<1|1];
        secv[k]=max(secv[k<<1],secv[k<<1|1]);
    }
    else if(maxv[k<<1]>maxv[k<<1|1]){
        maxv[k]=maxv[k<<1];
        maxc[k]=maxc[k<<1];
        secv[k]=max(secv[k<<1],maxv[k<<1|1]);
    }
    else{
        maxv[k]=maxv[k<<1|1];
        maxc[k]=maxc[k<<1|1];
        secv[k]=max(maxv[k<<1],secv[k<<1|1]);
    }
}
void push_tag(int k,int v){
    if(v>=maxv[k])return;
    sum[k]-=1LL*maxc[k]*(maxv[k]-v);
    maxv[k]=lazy[k]=v;
}
void push_down(int k){
    if(~lazy[k]){
        push_tag(k<<1,lazy[k]);
        push_tag(k<<1|1,lazy[k]);
        lazy[k]=-1;
    }
}
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R,lazy[k]=-1;
    int M=L+R>>1;
    if(L==R){
        sum[k]=maxv[k]=a[M];
        secv[k]=-1;
        maxc[k]=1;
        return;
    }
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
    push_up(k);
}
void update(int k,int L,int R,int v){
    if(maxv[k]<=v)return;
    if(L<=lef[k]&&rig[k]<=R&&secv[k]<v)
    return push_tag(k,v);
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
```

```
    if(mid>=L)update(k<<1,L,R,v);
    if(mid<R)update(k<<1|1,L,R,v);
    push_up(k);
}
int query_max(int k,int L,int R){
    if(L<=lef[k]&&rig[k]<=R)return maxv[k];
    push_down(k);
    int mid=lef[k]+rig[k]>>1,vl=-1,vr=-1;
    if(mid>=L)vl=query_max(k<<1,L,R);
    if(mid<R)vr=query_max(k<<1|1,L,R);
    return max(vl,vr);
}
LL query_sum(int k,int L,int R){
    if(L<=lef[k]&&rig[k]<=R)return sum[k];
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    LL s=0;
    if(mid>=L)s+=query_sum(k<<1,L,R);
    if(mid<R)s+=query_sum(k<<1|1,L,R);
    return s;
}
int main()
{
    int T=read_int();
    while(T--){
        int n=read_int(),q=read_int();
        _rep(i,1,n)a[i]=read_int();
        build(1,1,n);
        int opt,x,y;
        while(q--){
            opt=read_int(),x=read_int(),y=read_int();
            if(opt==0)
            update(1,x,y,read_int());
            else if(opt==1)
            enter(query_max(1,x,y));
            else
            enter(query_sum(1,x,y));
        }
    }
    return 0;
}
```

模板二

洛谷p6242

题意

给定一个序列，支持下述三种操作：

1. 对 $l\le i\le r$，$a_i=a_i+v$
2. 对 $l\le i\le r$，$a_i=\min(a_i,v)$
3. 询问区间 $[l,r]$ 的和
4. 询问区间 $[l,r]$ 的最大值
5. 询问区间 $[l,r]$ 的历史最大值

题解

先只考虑操作 $1\sim 4$，对区间加操作，直接按普通区间加处理即可。但注意区间加会修改操作最值操作的懒标记，最大值和次大值。

另外区间加需要特判最值操作的懒标记和区间次大值不存在的情况。区间加操作本身的复杂度显然为 $O(\log n)$。

考虑区间加操作对区间最值操作的影响。对某个结点，不难发现该结点的数集大小增加的最大可能值等于区间加过程中遍历的该结点的子节点数。

一个结点在一次区间加过程中最多有 $O(\log n)$ 个子结点被遍历，而一次区间加过程最多遍历 $O(\log n)$ 个结点。

所以一次区间加操作最多使数集总和增加 $O(\log^2 n)$，所以总时间复杂度变为 $O(n\log n+m\log^2 n)$。

```
const int MAXN=1e6+5,Inf=2e9;
int a[MAXN];
int lef[MAXN<<2],rig[MAXN<<2],maxv[MAXN<<2],maxc[MAXN<<2],secv[MAXN<<2];
int lazy1[MAXN<<2],lazy2[MAXN<<2];
LL sum[MAXN<<2];
void push_up(int k){
    sum[k]=sum[k<<1]+sum[k<<1|1];
    if(maxv[k<<1]==maxv[k<<1|1]){
        maxv[k]=maxv[k<<1];
        maxc[k]=maxc[k<<1]+maxc[k<<1|1];
        secv[k]=max(secv[k<<1],secv[k<<1|1]);
    }
    else if(maxv[k<<1]>maxv[k<<1|1]){
        maxv[k]=maxv[k<<1];
        maxc[k]=maxc[k<<1];
        secv[k]=max(secv[k<<1],maxv[k<<1|1]);
    }
    else{
        maxv[k]=maxv[k<<1|1];
        maxc[k]=maxc[k<<1|1];
        secv[k]=max(maxv[k<<1],secv[k<<1|1]);
    }
}
void push_add(int k,int v){
    sum[k]+=1LL*(rig[k]-lef[k]+1)*v;
```

```
    maxv[k]+=v,lazy1[k]+=v;
    if(secv[k]!=-Inf)secv[k]+=v;
    if(lazy2[k]!=-Inf)lazy2[k]+=v;
}
void push_min(int k,int v){
    if(v>=maxv[k])return;
    sum[k]-=1LL*maxc[k]*(maxv[k]-v);
    maxv[k]=lazy2[k]=v;
}
void push_down(int k){
    if(lazy1[k]){
        push_add(k<<1,lazy1[k]);
        push_add(k<<1|1,lazy1[k]);
        lazy1[k]=0;
    }
    if(lazy2[k]!=-Inf){
        push_min(k<<1,lazy2[k]);
        push_min(k<<1|1,lazy2[k]);
        lazy2[k]=-Inf;
    }
}
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R,lazy1[k]=0,lazy2[k]=-Inf;
    int M=L+R>>1;
    if(L==R){
        sum[k]=maxv[k]=a[M];
        secv[k]=-Inf;
        maxc[k]=1;
        return;
    }
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
    push_up(k);
}
void update_add(int k,int L,int R,int v){
    if(L<=lef[k]&&rig[k]<=R)
    return push_add(k,v);
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    if(mid>=L)update_add(k<<1,L,R,v);
    if(mid<R)update_add(k<<1|1,L,R,v);
    push_up(k);
}
void update_min(int k,int L,int R,int v){
    if(maxv[k]<=v)return;
    if(L<=lef[k]&&rig[k]<=R&&secv[k]<v)
    return push_min(k,v);
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    if(mid>=L)update_min(k<<1,L,R,v);
    if(mid<R)update_min(k<<1|1,L,R,v);
```

```
    push_up(k);
}
LL query_sum(int k,int L,int R){
    if(L<=lef[k]&&rig[k]<=R)return sum[k];
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    LL s=0;
    if(mid>=L)s+=query_sum(k<<1,L,R);
    if(mid<R)s+=query_sum(k<<1|1,L,R);
    return s;
}
int query_max(int k,int L,int R){
    if(L<=lef[k]&&rig[k]<=R)return maxv[k];
    push_down(k);
    int mid=lef[k]+rig[k]>>1,vl=-Inf,vr=-Inf;
    if(mid>=L)vl=query_max(k<<1,L,R);
    if(mid<R)vr=query_max(k<<1|1,L,R);
    return max(vl,vr);
}
int main()
{
    int n=read_int(),q=read_int();
    _rep(i,1,n)a[i]=read_int();
    build(1,1,n);
    while(q--){
        int opt=read_int(),l=read_int(),r=read_int();
        switch(opt){
            case 1:
                update_add(1,l,r,read_int());
                break;
            case 2:
                update_min(1,l,r,read_int());
                break;
            case 3:
                enter(query_sum(1,l,r));
                break;
            case 4:
                enter(query_max(1,l,r));
                break;
        }
    }
    return 0;
}
```

接下来考虑操作 $1,3,4,5$。发现只需要维护加法懒标记的历史最大值即可维护区间历史最大值。

接下来考虑操作 $1\sim 5$发现区间最值操作可以转化为对区间最值单独操作的特殊区间加法。

于是考虑对每个结点维护区间最值懒标记，区间最值的历史最值懒标记，区间非最值懒标记，区间非最值的历史最值懒标记。

总时间复杂度仍然为 $O(n\log n+m\log^2 n)$

```
const int MAXN=1e6+5,Inf=2e9;
int a[MAXN];
int
lef[MAXN<<2],rig[MAXN<<2],maxv[MAXN<<2],mmaxv[MAXN<<2],maxc[MAXN<<2],secv[M
AXN<<2];
int lazy1[MAXN<<2],lazy2[MAXN<<2],mlazy1[MAXN<<2],mlazy2[MAXN<<2];
LL sum[MAXN<<2];
void push_up(int k){
    sum[k]=sum[k<<1]+sum[k<<1|1];
    mmaxv[k]=max(mmaxv[k<<1],mmaxv[k<<1|1]);
    if(maxv[k<<1]==maxv[k<<1|1]){
        maxv[k]=maxv[k<<1];
        maxc[k]=maxc[k<<1]+maxc[k<<1|1];
        secv[k]=max(secv[k<<1],secv[k<<1|1]);
    }
    else if(maxv[k<<1]>maxv[k<<1|1]){
        maxv[k]=maxv[k<<1];
        maxc[k]=maxc[k<<1];
        secv[k]=max(secv[k<<1],maxv[k<<1|1]);
    }
    else{
        maxv[k]=maxv[k<<1|1];
        maxc[k]=maxc[k<<1|1];
        secv[k]=max(maxv[k<<1],secv[k<<1|1]);
    }
}
void push_tag(int k,int v1,int v2,int mv1,int mv2){
    sum[k]+=1LL*maxc[k]*v1+1LL*(rig[k]-lef[k]+1-maxc[k])*v2;
    mmaxv[k]=max(mmaxv[k],maxv[k]+mv1);
    mlazy1[k]=max(mlazy1[k],lazy1[k]+mv1);
    maxv[k]+=v1,lazy1[k]+=v1;
    mlazy2[k]=max(mlazy2[k],lazy2[k]+mv2);
    if(secv[k]!=-Inf)secv[k]+=v2;
    lazy2[k]+=v2;
}
void push_down(int k){
    int temp=max(maxv[k<<1],maxv[k<<1|1]);
    if(temp==maxv[k<<1])
    push_tag(k<<1,lazy1[k],lazy2[k],mlazy1[k],mlazy2[k]);
    else
    push_tag(k<<1,lazy2[k],lazy2[k],mlazy2[k],mlazy2[k]);
    if(temp==maxv[k<<1|1])
    push_tag(k<<1|1,lazy1[k],lazy2[k],mlazy1[k],mlazy2[k]);
    else
    push_tag(k<<1|1,lazy2[k],lazy2[k],mlazy2[k],mlazy2[k]);
    lazy1[k]=lazy2[k]=mlazy1[k]=mlazy2[k]=0;
}
void build(int k,int L,int R){
```

```
    lef[k]=L,rig[k]=R,lazy1[k]=lazy2[k]=mlazy1[k]=mlazy2[k]=0;
    int M=L+R>>1;
    if(L==R){
        sum[k]=maxv[k]=mmaxv[k]=a[M];
        secv[k]=-Inf;
        maxc[k]=1;
        return;
    }
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
    push_up(k);
}
void update_add(int k,int L,int R,int v){
    if(L<=lef[k]&&rig[k]<=R)
    return push_tag(k,v,v,v,v);
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    if(mid>=L)update_add(k<<1,L,R,v);
    if(mid<R)update_add(k<<1|1,L,R,v);
    push_up(k);
}
void update_min(int k,int L,int R,int v){
    if(maxv[k]<=v)return;
    if(L<=lef[k]&&rig[k]<=R&&secv[k]<v)
    return push_tag(k,v-maxv[k],0,v-maxv[k],0);
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    if(mid>=L)update_min(k<<1,L,R,v);
    if(mid<R)update_min(k<<1|1,L,R,v);
    push_up(k);
}
LL query_sum(int k,int L,int R){
    if(L<=lef[k]&&rig[k]<=R)return sum[k];
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    LL s=0;
    if(mid>=L)s+=query_sum(k<<1,L,R);
    if(mid<R)s+=query_sum(k<<1|1,L,R);
    return s;
}
int query_max(int k,int L,int R){
    if(L<=lef[k]&&rig[k]<=R)return maxv[k];
    push_down(k);
    int mid=lef[k]+rig[k]>>1,vl=-Inf,vr=-Inf;
    if(mid>=L)vl=query_max(k<<1,L,R);
    if(mid<R)vr=query_max(k<<1|1,L,R);
    return max(vl,vr);
}
int query_mmax(int k,int L,int R){
    if(L<=lef[k]&&rig[k]<=R)return mmaxv[k];
```

```
    push_down(k);
    int mid=lef[k]+rig[k]>>1,vl=-Inf,vr=-Inf;
    if(mid>=L)vl=query_mmax(k<<1,L,R);
    if(mid<R)vr=query_mmax(k<<1|1,L,R);
    return max(vl,vr);
}
int main()
{
    int n=read_int(),q=read_int();
    _rep(i,1,n)a[i]=read_int();
    build(1,1,n);
    while(q--){
        int opt=read_int(),l=read_int(),r=read_int();
        switch(opt){
            case 1:
                update_add(1,l,r,read_int());
                break;
            case 2:
                update_min(1,l,r,read_int());
                break;
            case 3:
                enter(query_sum(1,l,r));
                break;
            case 4:
                enter(query_max(1,l,r));
                break;
            case 5:
                enter(query_mmax(1,l,r));
                break;
        }
    }
    return 0;
}
```

算法练习

习题一

题意

给定一个长度为 n 序列，支持下列操作：

1. 询问区间 $[l,r]$ 的最大值
2. 询问区间 $[l,r]$ 的历史最大值
3. 对 $i\in [l,r],a_i=a_i+v$
4. 对 $i\in [l,r],a_i=v$

题解

对加法操作，考虑维护区间当前加法懒标记和区间历史最大加法懒标记。对赋值操作，考虑维护区间当前赋值懒标记和区间历史最大赋值懒标记。

考虑标记下放的本质，即将当前区间的所有操作序列添加到子区间操作序列的后面，同时维护该操作对答案的影响。一个策略为合并操作。

发现将加法操作添加到加法操作后面可以直接叠加，可以合并为一个加法操作。

将赋值操作添加到加法操作无法合并，留下一个赋值操作。

将赋值操作添加到赋值操作后面直接覆盖即可，可以合并为一个赋值操作。

将加法操作添加到赋值操作后面可以理解为更新上一次的赋值操作，最后合共为一个赋值操作。

于是整个操作序列最后可以合并为一个加法操作和一个赋值操作。懒标记下放的同时维护历史最值即可，时间复杂度 $O(n+q\log n)$。

```
int a[MAXN];
int lef[MAXN<<2],rig[MAXN<<2],maxv[MAXN<<2],mmaxv[MAXN<<2];
int
lazy1[MAXN<<2],lazy2[MAXN<<2],mlazy1[MAXN<<2],mlazy2[MAXN<<2],has_set[MAXN<
<2];
void push_up(int k){
    maxv[k]=max(maxv[k<<1],maxv[k<<1|1]);
    mmaxv[k]=max(mmaxv[k<<1],mmaxv[k<<1|1]);
}
void push_set(int k,int v,int mv){
    maxv[k]=v,mmaxv[k]=max(mmaxv[k],mv);
    if(has_set[k]){
        lazy2[k]=v;
        mlazy2[k]=max(mlazy2[k],mv);
    }
    else{
        has_set[k]=1;
        lazy2[k]=v;
        mlazy2[k]=mv;
    }
}
void push_add(int k,int v,int mv){
    if(has_set[k])return push_set(k,lazy2[k]+v,lazy2[k]+mv);
    mmaxv[k]=max(mmaxv[k],maxv[k]+mv);
    mlazy1[k]=max(mlazy1[k],lazy1[k]+mv);
    maxv[k]+=v,lazy1[k]+=v;
}
void push_down(int k){
    push_add(k<<1,lazy1[k],mlazy1[k]);
    push_add(k<<1|1,lazy1[k],mlazy1[k]);
    lazy1[k]=mlazy1[k]=0;
    if(has_set[k]){
```

```
        push_set(k<<1,lazy2[k],mlazy2[k]);
        push_set(k<<1|1,lazy2[k],mlazy2[k]);
        has_set[k]=0;
    }
}
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R,lazy1[k]=mlazy1[k]=has_set[k]=0;
    int M=L+R>>1;
    if(L==R){
        maxv[k]=mmaxv[k]=a[M];
        return;
    }
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
    push_up(k);
}
void update_add(int k,int L,int R,int v){
    if(L<=lef[k]&&rig[k]<=R)
    return push_add(k,v,v);
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    if(mid>=L)update_add(k<<1,L,R,v);
    if(mid<R)update_add(k<<1|1,L,R,v);
    push_up(k);
}
void update_set(int k,int L,int R,int v){
    if(L<=lef[k]&&rig[k]<=R)
    return push_set(k,v,v);
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    if(mid>=L)update_set(k<<1,L,R,v);
    if(mid<R)update_set(k<<1|1,L,R,v);
    push_up(k);
}
int query_max(int k,int L,int R){
    if(L<=lef[k]&&rig[k]<=R)return maxv[k];
    push_down(k);
    int mid=lef[k]+rig[k]>>1,vl=-Inf,vr=-Inf;
    if(mid>=L)vl=query_max(k<<1,L,R);
    if(mid<R)vr=query_max(k<<1|1,L,R);
    return max(vl,vr);
}
int query_mmax(int k,int L,int R){
    if(L<=lef[k]&&rig[k]<=R)return mmaxv[k];
    push_down(k);
    int mid=lef[k]+rig[k]>>1,vl=-Inf,vr=-Inf;
    if(mid>=L)vl=query_mmax(k<<1,L,R);
    if(mid<R)vr=query_mmax(k<<1|1,L,R);
    return max(vl,vr);
}
int main()
```

```
{
    int n=read_int();
    _rep(i,1,n)a[i]=read_int();
    build(1,1,n);
    int q=read_int();
    while(q--){
        char opt=get_char();
        int l=read_int(),r=read_int();
        switch(opt){
            case 'Q':
                enter(query_max(1,l,r));
                break;
            case 'A':
                enter(query_mmax(1,l,r));
                break;
            case 'P':
                update_add(1,l,r,read_int());
                break;
            case 'C':
                update_set(1,l,r,read_int());
                break;
        }
    }
    return 0;
}
```

习题二

2020牛客国庆集训派对day1 B题

题意

给定序列 A，定义 $G(i,j)=\text{gcd}(a_i,a_{i+1}\cdots a_j),M(i,j)=\max(a_i,a_{i+1}\cdots a_j)$，求

$$ \sum_{i=1}^n\sum_{j=i}^nG(i,j)M(i,j) $$

题解

考虑计算 $\sum_{j=1}^iG(i,j)M(i,j)$ 的贡献。

发现 $1\le j\le i$ 的 $G(i,j)$ 只有 $O(\log v)$ 个取值，同时有 $G(i,j)=\text{gcd}(G(i-1,j),a_i)$，

于是可以 $O(\log n\log v)$ 维护所有不同的 $G(i,j)$，不同取值的 $G(i,j)$ 将 $[1,i]$ 划分为若干区间。

又有 $M(i,j)=\max (M(i-1,j),a_i)$，于是可以吉司机线段树维护区间最值操作下的区间和。

对每个被划分的区间直接查询求和即可，时间复杂度 $O(\log n\log v)$，于是总时间复杂 $O(n\log n\log

v)$

```
const int MAXN=2e5+5,Mod=1e9+7;
int a[MAXN];
int
lef[MAXN<<2],rig[MAXN<<2],minv[MAXN<<2],minc[MAXN<<2],secv[MAXN<<2],lazy[MA
XN<<2];
LL sum[MAXN<<2];
void push_up(int k){
    sum[k]=(sum[k<<1]+sum[k<<1|1])%Mod;
    if(minv[k<<1]==minv[k<<1|1]){
        minv[k]=minv[k<<1];
        minc[k]=minc[k<<1]+minc[k<<1|1];
        secv[k]=min(secv[k<<1],secv[k<<1|1]);
    }
    else if(minv[k<<1]<minv[k<<1|1]){
        minv[k]=minv[k<<1];
        minc[k]=minc[k<<1];
        secv[k]=min(secv[k<<1],minv[k<<1|1]);
    }
    else{
        minv[k]=minv[k<<1|1];
        minc[k]=minc[k<<1|1];
        secv[k]=min(minv[k<<1],secv[k<<1|1]);
    }
}
void push_tag(int k,int v){
    if(v<=minv[k])return;
    sum[k]=(sum[k]+1LL*minc[k]*(v-minv[k]))%Mod;
    minv[k]=lazy[k]=v;
}
void push_down(int k){
    if(~lazy[k]){
        push_tag(k<<1,lazy[k]);
        push_tag(k<<1|1,lazy[k]);
        lazy[k]=-1;
    }
}
void build(int k,int L,int R){
    lef[k]=L,rig[k]=R,lazy[k]=-1;
    if(L==R){
        sum[k]=minv[k]=0;
        secv[k]=2e9;
        minc[k]=1;
        return;
    }
    int M=L+R>>1;
    build(k<<1,L,M);
    build(k<<1|1,M+1,R);
    push_up(k);
```

```
}
void update(int k,int L,int R,int v){
    if(minv[k]>=v)return;
    if(L<=lef[k]&&rig[k]<=R&&secv[k]>v)
    return push_tag(k,v);
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    if(mid>=L)update(k<<1,L,R,v);
    if(mid<R)update(k<<1|1,L,R,v);
    push_up(k);
}
int query_sum(int k,int L,int R){
    if(L<=lef[k]&&rig[k]<=R)return sum[k];
    push_down(k);
    int mid=lef[k]+rig[k]>>1;
    LL s=0;
    if(mid>=L)s+=query_sum(k<<1,L,R);
    if(mid<R)s+=query_sum(k<<1|1,L,R);
    return s%Mod;
}
int gcd(int a,int b){
    int t;
    while(b){
        t=b;
        b=a%b;
        a=t;
    }
    return a;
}
struct Node{
    int v,pos;
}Stack[MAXN],temp[MAXN];
int main()
{
    int n=read_int();
    build(1,1,n);
    _rep(i,1,n)
    a[i]=read_int();
    int top=0;
    Stack[top]=Node{0,0};
    int ans=0;
    _rep(i,1,n){
        update(1,1,i,a[i]);
        _rep(j,1,top)
        Stack[j].v=gcd(Stack[j].v,a[i]);
        Stack[++top]=Node{a[i],i};
        _rep(i,1,top)
        temp[i]=Stack[i];
        int top2=top;
        top=0;
```

```
        _rep(i,1,top2){
            while(temp[i].v==Stack[top].v)top--;
            Stack[++top]=temp[i];
        }
        for(int i=top;i;i--)
ans=(ans+1LL*Stack[i].v*query_sum(1,Stack[i-1].pos+1,Stack[i].pos))%Mod;
    }
    enter(ans);
    return 0;
}
```