线段树练习

例1 矩形面积并（扫描线法）

二维平面上有 n 个矩形，给出第 i 个矩形的左上角和右下角坐标（x1[i], y1[i]), (x2[i], y2[i]).
求这些矩形的并的面积。

n<=1e5, 0<x, y<3e4.

sol

1、考虑二维差分矩阵，每个矩形 O(1) 打标记，还原时可以计算出每个点覆盖几次，计算时只计算一次即可。

时间复杂度 $O(n+x^2)$。

sol

直接考虑二维不方便，我们可以枚举第一维，然后统计第二维上的信息。

这也就是扫描线的思想。

枚举 y, 对于每个 y，统计 x 坐标上哪些被矩形覆盖（即算出多少个区间可以统计入答案）。

具体的，对每个矩形上边和下边进行排序，然后从小到大枚举y，碰到下边则需要给 [xi, yi] 区间+1，碰到下边的话对该区间减 1.

最后需要统计 [1, 3e4] 上有几个数 > 0.

sol

因为计算大于 0 的个数不方便，注意到 [1, 3e4] 上的数都是非负的。

因为没有点被覆盖了负数次。

因此我们可以统计 0 的个数。也就是最小值的个数，然后用 3e4 减去 0 的个数，就得到了被覆盖的点的个数。

因此，这道题就变成了维护一个支持以下两种操作的数据结构：

1、区间+1或-1.

2、统计一个区间最小值个数。

时间复杂度 O(nlogn).

extra 矩形面积并加强版（扫描线法）

二维平面上有 n 个矩形，给出第 i 个矩形的左上角和右下角坐标（x1[i], y1[i]), (x2[i], y2[i]).

求这些矩形的并的面积。

n<=1e5, 0<x, y<1e9.

sol

由于坐标范围太大，没法直接建 10^9 的线段树。

离散化。

注意计算答案需要用原来的 x, y.

核心代码

```cpp
#define mid (l+r>>1)
#define lc x<<1
#define rc x<<1|1
int xbin[N],ybin[N];
int xcnt,ycnt,n;
int cnt[N],minn[N],tag[N];
struct rect{
    int x1,y1,x2,y2;
}r[N];
struct node{
    int x1,x2,v;
};
vector<node> g[N];
```

```cpp
void pushup(int k){
    if(minn[lc]==minn[rc]){
        minn[k]=minn[lc];
        cnt[k]=cnt[lc]+cnt[rc];
    }else{
        int f=(minn[lc]<minn[rc])?lc:rc;
        minn[k]=f;
        cnt[k]=cnt[f];
    }
}
void pushdown(int k){
    if(tag[k]!=0){
        minn[lc]+=tag[k],minn[rc]+=tag[k];
        tag[lc]+=tag[k],tag[rc]+=tag[k];
        tag[k]=0;
    }
}
void change(int k,int l,int r,int from,int to,int v){
    if(from<=l&&r<=to){
        tag[k]+=v,minn[k]+=v;
        return;
    }
    pushdown(k);
    if(from<=mid) change(lc,l,mid,from,to,v);
    if(mid<to) change(rc,mid+1,r,from,to,v);
    pushup(k);
}
```

```cpp
void build(int k,int l,int r){
    minn[k]=tag[k]=0;
    if(l==r){
        cnt[k]=xbin[l]-xbin[l-1];
        return;
    }
    build(lc,l,mid);
    build(rc,mid+1,r);
    pushup(k);
}
```

核心代码

```cpp
n=read();
for(int i=1;i<=n;i++){
    r[i],x1=read(),r[i].y1=read(),r[i].x2=read(),r[i].y2=read();
    xbin[++xcnt]=r[i].x1,xbin[++xcnt]=r[i].x2;
    ybin[++ycnt]=r[i].y1,ybin[++ycnt]=r[i].y2;
}
sort(xbin+1,xbin+xcnt+1);
sort(ybin+1,ybin+ycnt+1);
xcnt=unique(xbin+1,xbin+xcnt+1)-xbin-1;
ycnt=unique(ybin+1,ybin+ycnt+1)-ybin-1;
for(int i=1;i<=n;i++){
    int x1=lower_bound(xbin+1,xbin+xcnt+1,r[i].x1)-xbin;
    int x2=lower_bound(xbin+1,xbin+xcnt+1,r[i].x2)-xbin;
    int y1=lower_bound(ybin+1,ybin+ycnt+1,r[i].y1)-ybin;
    int y2=lower_bound(ybin+1,ybin+ycnt+1,r[i].y2)-ybin;
    if(x1<x2){
        g[y1].push_back((node){x1+1,x2,1});
        g[y2].push_back((node){x1+1,x2,-1});
    }
}
```

```cpp
xbin[0]=xbin[1];
ybin[0]=ybin[1];
build(1,1,xcnt);
int ans=0;
for(int i=1;i<=ycnt;i++){
    int tmp=cnt[1];
    if(minn[1]>0) tmp=0;
    ans+=(ybin[i]-ybin[i-1])*(xbin[xcnt]-xbin[1]-tmp);
    for(int j=0;j<g[i].size();j++){
        change(1,1,xcnt,g[i][j].x1,g[i][j].x2,g[i][j].v);
    }
}
cout<<ans;
```

例2 hdu3333（线段树离线查询）

- 一个长度为 n 的序列，q 次询问，每次询问 (x, y)：求出 [x, y]区间中出现过的数字之和。（即重复数字只算一次)

- n<=3e4, q<=1e5

sol

- 考虑直接用线段树维护，发现无法合并左右两个子节点，因为不能统计每个节点出现了哪几种数。

- 解法一：暴力。
- 解法二：设 left[i] 为 a[i] 左边第一次与 a[i] 相等的下标，则 [x, y] 区间里若 left[i] < x 则可以累加进答案。
- 预处理 left 数组可以离散化后 left[i]=last[a[i]],last[a[i]]=i;

sol

- 问题变成每次求 [x,y] 中 left<x 的 a 的和。
- 这个时候考虑离线求解。

- 不用每次询问求解，先对left值进行排序，然后顺序枚举 x, 依次把 满足 left[i]<x 的 a[i] 插入线段树中的第 i 个位置。
- 即：对于任意 y, 询问 [x, y] 的话，如果 i 在 [x, y] 中，那 a[i] 统计入答案。
- 时间复杂度O((n+q) lgn)

核心代码

```
cin>>n;
for(int i=1;i<=n;i++) cin>>a[i],bin[++cnt]=a[i];
cin>>m;
for(int i=1;i<=m;i++) cin>>q[i].l>>q[i].r,q[i].id=i;
sort(bin+1,bin+cnt+1);
cnt = unique(bin+1,bin+cnt+1)-bin-1;
for(int i=1;i<=n;i++) a[i]=lower_bound(bin+1,bin+cnt+1,a[i])-bin;
```

```
memset(last,0,sizeof(last));
for(int i=1;i<=n;i++){
    left[i]=last[a[i]];
    p[i]=(node){left[i],i,bin[a[i]]};
    last[a[i]]=i;
}
sort(p+1,p+n+1,cmp);
sort(q+1,q+m+1,cmp);
memset(sum,0,sizeof(sum));
int j=1;
for(int i=1;i<=m;i++){
    while(j<=n&&p[j].l < q[i].l){
        add(p[j].r,p[j].id,1,n,1);
        j++;
    }
    ans[q[i].id] = query(q[i].l,q[i].r,1,n,1);
}
for(int i=1;i<=m;i++) cout<<ans[i]<<endl;
```

例3 poj 2828 （线段树上二分）

- 有 n 个人来排队，第 i 个人来的时候会排在 p[i] (0<=p[i]<=i) 个人后面，同时它会被分配一个数字 v[i]。
- 现在告诉你 n 对（p[i], v[i]），最后按照队伍顺序输出每个人的数字
- n<=2e5

【样例输入】

4

0 77

1 51

1 33

2 69

【样例输出】

77 33 69 51

sol

- 考虑模拟做法
- 数组和指针维护都是O（n^2）.

sol

正难则反。

倒着考虑每个人，就可以确定每个人的位置。

比如第 n 个人，一定在 p[n]+1 位置上。

然后考虑如何放第 n-1 个人的位置。

如果 p[n-1]+1 < p[n]+1, 也就是说第 n-1 个人的位置没有因为第 n 个人的进入而改变，因此可以放在 p[n-1]+1 处。

如果 p[n-1]+1 >= p[n]+1, 也就是说第 n-1 个人在第 n 个人进入后向后移了一位，因此第 n-1 个人的位置应在 p[n-1]+2 处。

我们可以总结为第 n-1 个人的位置在除了第 n 个人的位置外的第 p[n-1]+1 处。

sol

接着考虑第 n-2 个人的位置，可以分析得应该在除了第 n 个人 和第 n-1 个人外的 p[n-2]+1处。

因此我们发现，我们如果只考虑相对位置，这个问题是个规模逐渐减小的递归问题。
问题规模为 n 时，确定第 n 个人的位置，删除这个位置后，问题变成 n-1 规模的同样的问题，与 n 无关。只是在最后输出需要考虑绝对位置，留出 n 所在的空位。

于是问题就变成了: 一开始有 n 个位置，每次查询第 p[i]+1 个空位置，然后将该位置变成非空。

sol

一个简单的想法就是用线段树维护一个全 1 的数组 a。

查找第 x 个空位置就是找一个下标 i 使得 a[1…i] 的前缀和等于 x。

由于 a 是单增的，所以我们可以二分下标 i。

用线段树维护区间和，每次二分时查询即可。

时间复杂度为 O(n logn logn)

sol

注意线段树本身就有分治的性质。

使用分治套分治时我们要思考能否减少冗余计算，只使用一层分治解决问题。

于是我们可以直接在线段树上二分。

核心代码

```
int query(int x,int l,int r,int k){
    if(l==r){
        sum[k]=0;
        return l;
    }
    int mid=(l+r)>>1,ret=0;
    if(x<=sum[lc]) ret=query(x,l,mid,lc);
    else ret=query(k-sum[lc],mid+1,r,rc);
    update(k);
    return ret;
}
```

```
cin>>n;
for(int i=1;i<=n;i++) cin>>pos[i]>>val[i];
build(1,n,1);
for(int i=n;i>=1;i--){
    int k=find(pos[i]+1,1,n,1);
    ans[k]=val[i];
}
for(int i=1;i<=n;i++) cout<<ans[i];
```

例4 hdu 1540 （线段树上找答案）

- 有 n 个连在一起的地道，接下来有 m 个操作，每个操作有以下几种：

D x : 炸掉 x 号地道

Q x : 查询包含 x 的最长地道长度并输出

R : 修复上一个被炸的地道

- n,m<=50000

【样例输入】

```
7 9
D 3
D 6
D 5
Q 4
Q 5
R
Q 4
R
Q 4
```

【样例输出】

```
1
0
2
4
```

sol

用线段树来维护每个点是否被炸掉，0 表示炸掉，1表示完好。

考虑如何查询包含 x 的最大连续长度。

我们先放宽条件，思考如何查询 [1, n]里的最大长度。

sol

假设我们用一个 f[k] 来维护编号为 k 的区间的最长连续长度。
考虑怎样合并两个子节点的信息。
此时我们发现:
不能通过 f[lc]+f[rc] 来更新 f[k],因为连续段可能没在一起。
不能通过 max(f[lc], f[rc]) 来更新 f[k],因为连续段可能连在一起。
因此，对于每个区间，我们需要再维护两个值:
lmax[k]: 区间最左端最长的连续段长度
rmax[k]:区间最右端最长的连续段长度
此时，f[k] 可以用 rmax[lc]+lmax[rc]来更新。
更新时 lmax, rmax 也要跟着更新。

sol

然后回到本问题，现在要求包含 x 的最长连续字段。

假设我们现在在编号为 k ，区间为 [l, r] 。

我们首先需要判断包含 x 的最长连续字段是否横跨了 mid:

如果 mid-rmax[lc]+1 <= x <=mid, 此时返回 rmax[lc]+lmax[rc]即可。

如果 mid+1 <= x <= mid+lmax[rc], 此时返回 rmax[lc]+lmax[rc]即可。

否则说明 x 到 mid 中间肯定有一个点被炸掉了。这样另一个子节点对答案没影响。

那么递归查询 k 所在的子节点就行。

核心代码

```
void update(int l,int r,int k){
    f[k]=max(max(f[lc],f[rc]),rmax[lc]+lmax[rc]);
    int mid=(l+r)>>1;
    lmax[k]=lmax[lc];
    if(lmax[lc]==mid-l+1) lmax[k]+=lmax[rc];
    rmax[k]=rmax[rc];
    if(rmax[rc]==r-mid) rmax[k]+=rmax[lc];
}
```

```
int query(int x,int l,int r,int k){
    if(l==r) return f[k];
    int mid=(l+r)>>1;
    if(x<=mid){
        if(mid-rmax[lc]<=k) return rmax[lc]+lmax[rc];
        else return query(x,l,mid,lc);
    }else{
        if(k<=mid+lmax[rc]) return rmax[lc]+lmax[rc];
        else return query(x,mid+1,r,rc);
    }
}
```

```
void change(int pos,int v,int l,int r,int k){
    if(l==r){
        lmax[k]=rmax[k]=f[k]=v;
        return;
    }
    int mid=(l+r)>>1;
    if(pos<=mid) change(pos,v,l,mid,lc);
    else change(pos,v,mid_1,r,rc);
    update(l,r,k);
}
```

```
    cin>>n>>m;
    build(1,n,1);
    char op[2];
    int x,top=0;
    while(m--){
        cin>>op;
        if(op[0]=='R'){
            change(s[top],1,1,n,1);
            top--;
        }else if(op[0]=='D'){
            cin>>x;
            change(x,0,1,n,1);
            s[++top]=x;
        }else{
            cin>>x;
            cout<<query(x,1,n,1)<<endl;
        }
    }
```

例5 cf#406div1 b Legacy（线段树优化建图）

一张图上有n个点，现在有m次操作。每次操作有3种:

1 a b c：从 a 到 b 连一条权值为 c 的边。

2 a b c d：从 a 到 [b, c] 中的每个点连一条权值为 d 的单向边。

3 a b c d：从 [b, c] 中的每个点向 a 连一条权值为 d 的单向边。

最后给你一个起点，求起点到其他点的最短路长度。

n,m <=1e5

sol

考虑暴力做法，如果从 a 向 [b, c] 每个点连边，总共有 n^2 条边，接近完全图。

问题在于如何优化区间建图。

通过类似区间询问的方法在线段树上最多 logn 个区间连边。边数级别为 O(mlogn)，点数级别为 O(n)。

最后在新建的图上跑堆优化的 dj，时间复杂度 O((n+m) log^2 n).

核心代码

```
void addedge(int x,int y,int w,int flag);
//flag=0 x->y; flag=1 y->x;

void build(int l,int r,int x,int flag){
    if(l==r){
        id[flag][x]=l;return;
    }
    id[flag][x]= ++tot;
    int mid=(l+r)>>1;
    build(l,mid,lc,flag);
    build(mid+1,r,rc,flag);
    addedge(id[flag][x],id[flag][lc],0,flag);
    addedge(id[flag][x],id[flag][rc],0,flag);
}
```

例6 poj3321 Apple Tree（线段树维护树上信息）

有一颗以 1 为根的树，每个节点上都有一个苹果。

每次有以下两种操作:

C x：x节点上的苹果发生了改变。如果原来有，则变为没有；如果没有，则变为有。

Q x：询问以 x 为根的子树总共有多少苹果。

n,m <=1e5

sol

线段树维护的是区间的信息。
现在询问的是子树上的信息。
如何把子树上的点转化为一个区间呢?

使用 dfs 序，一个子树对应着一个区间。
方法：dfs时进入某个节点 u 记一个时间戳 start[u]，等回溯回来再记一个时间戳 end[u]。
这样修改时就直接 change(start[u],1,n,1), 查询时输出 query(start[u],end[u],1,n,1).

核心代码

```
void dfs(int u,int fa){
    start[u]=++tot;
    for(int i=h[u];i;i=e[i].next){
        int v=e[i].v;
        if(v==fa) continue;
        dfs(v,u);
    }
    end[u]=tot;
}
void change(int pos,int l,int r,int x){
    if(l==r){
        sum[x]^=1;return;
    }
    int mid=(l+r)>>1;
    if(pos<=mid) change(pos,l,mid,lc);
    else change(pos,mid+1,r,rc);
    update(x);
}
```

例6extra poj3321 Apple Tree

有一颗以 1 为根的树，每个节点上都有一个苹果。

每次有以下两种操作：

C x：x节点上的苹果发生了改变。如果原来有，则变为没有；如果没有，则变为有。

Q x：询问和 x 节点相邻的节点共有多少苹果。

n,m <=1e5

sol

使用 bfs 序。

一个点的相邻节点对应 bfs 序列上的一段+这个节点的父亲。

例7 poj2777 Count Color（线段树维护区间可合并信息）

对区间 [1, n] 进行两种操作:

1、 C a b t： 区间 [a, b] 染色为 t。

2、 P a b： 查询区间 [a, b]中有多少种不同颜色

n,m <=1e5， 1<=t<=30.

样例输入:

2 2 4
C 1 1 2
P 1 2
C 2 2 2
P 1 2

样例输出:

2
1

sol

用线段树来维护一个区间有哪几种颜色。
看到 t 的数据范围最多只有30，考虑类似状压，用二进制为来表示颜色。
每个区间维护一个mask，若 mask 的第 i 位为 1，则这个区间有第 i 种颜色。

查询的时候得到这个区间的mask，统计有几个 1 即可。

核心代码

```
void pushup(int x){
    mask[x] = mask[lc]|mask[rc];
}
void pushdown(int x){
    if(tag[x]!=0){
        tag[lc]=tag[rc]=tag[x];
        mask[lc]=mask[rc]=1<<tag[x];
        tag[x]=0;
    }
}
void build(int l,int r,int x){
    if(l==r){
        mask[x]=1;
        return;
    }
    int mid=(l+r)>>1;
    build(l,mid,lc);
    build(mid+1,r,rc);
    pushup(x);
}
```

```
void change(int from,int to,int t,int l,int r,int x){
    if(from<=l&&r<=to){
        mask[x]=1<<t;
        tag[x]=t;
        return;
    }
    pushdown(x);
    int mid=(l+r)>>1;
    if(from<=mid) change(from,to,t,l,mid,lc);
    if(mid<to) change(from,to,t,mid+1,r,rc);
    update(x);
}
int query(int from,int to,int l,int r,int x){
    if(from<=l&&r<=to) return mask[x];
    pushdown(x);
    int mid=(l+r)>>1,ret=1;
    if(from<=mid) ret|= query(from,to,l,mid,lc);
    if(mid<to) ret|= query(from,to,mid+1,r,rc);
    return ret;
}
```

sol

常数优化:

bitcount[x] 表示 x 的二进制中 1 的个数，只需开2^16大小

```
bitcount[1]=1;
for(int i=2;i<=(1<<16);i++){
    bitcount[i]=bitcount[i>>1]+(i&1);
}
ans=query(a,b,1,n,1);
bitcount[ans>>16]+bitcount[ans&((1<<16)-1)];
```

例8 spoj gss4 （线段树维护区间不可合并信息）

一个长为 n 的序列 A，里面每个数都是正数，且总和小于等于 10^18，有 m 个操作，每个操作有2种：

1 x y：把 [x, y] 区间的数全部开方（向下取整）

2 x y：询问 [x, y]的区间和。

n,m <=1e5。

sol

线段树维护。

难点在于不好去维护区间开方的标记，即对某个区间打上标记后无法在O(1)知道这个区间中的数开方后的总和。

这个时候，我们可以不再追求每次的时间复杂度都保持在O(logn)，而是考虑均摊复杂度。

注意到只有区间开方，没有要求区间修改，也就是每个数都在不断变小。

10^18，最多开7次就会变为1。

因此，我们只需要对区间里的数暴力修改，每个数最多都会被修改7次，这部分对整个时间复杂度的贡献是 O(n) 的。

sol

如果发现区间全都是1，则不用修改。

考虑维护一个区间最大值，如果最大值不是1，那么继续递归进行暴力修改。

如果最大值是 1，则终止。

这样一来，虽然某个操作需要暴力修改 n 个数，时间复杂度会达到O(n)。

但是总的来看，每个数都只会保留改 7 次，因此均摊下来时间复杂度为O(7nlogn)。

核心代码

```
void pushup(int x){
    sum[x]=sum[lc]+sum[rc];
    maxx[x]=max(maxx[lc],maxx[rc]);
}
void build(int l,int r,int x){
    if(l==r){
        sum[x]=maxx[x]=a[l];
        return;
    }
    int mid=(l+r)>>1;
    build(l,mid,lc);
    build(mid+1,r,rc);
    pushup(x);
}
ll query(int a,int b,int l,int r,int x){
    if(a<=l&&r<=b) return sum[x];
    int mid=(l+r)>>1;
    ll ret=0;
    if(a<=mid) ret+=query(a,b,l,mid,lc);
    if(mid<b) ret+=query(a,b,mid+1,r,rc);
    return ret;
}
```

```
void change(int a,int b,int l,int r,int x){
    if(l==r){
        sum[x]=maxx[x]=(ll)(floor(sqrt((double)sum[x])));
        return;
    }
    int mid=(l+r)>>1;
    if(a<=mid&&maxx[lc]>1) change(a,b,l,mid,lc);
    if(mid<b&&maxx[rc]>1) change(a,b,mid+1,r,rc);
    pushup(x);
}
```

例8extra spoj gss4 （线段树维护区间不可合并信息）

一个长为 n 的序列 A，里面每个数都是正数，且总和小于等于 10^18，有 m 个操作，每个操作有2种：

1 x y m：把 [x, y] 区间的数都对m取模。

2 x y：询问 [x, y]的区间和。

n,m <=1e5。

sol

线段树维护。

取模操作也不能打标记来结局。

同样考虑均摊复杂度，每次对 m 取模，每个数也会不断变小。

每个数至少会变成原来一半或者不变。

因此每个数最多变化log次。

修改操作即每次对区间大于m的数进行暴力修改。

同理，对每个区间记录一个最大值。如果大于m，需要暴力修改，否则终止。

总的时间复杂度 O(n*logn*logv)

例9 spoj gss1 （线段树维护区间子段和）

一个长为 n 的序列 A， m次询问：

x y: 输出 max{ A[i]+...A[j] },其中(x<=i<=j<=y)。即 [x, y]区间最大子段和(连续）。

| a[i] |<=15007, n,m <=50000。

sol

线段树维护每个区间的答案。

maxx[x] 表示区间 x 的答案。maxx[x] 不能由 maxx[lc] 和 maxx[rc] 直接合并。

类似于炸地道那道题，我们还需要记录每个区间的前缀和最大值 lmax[x]和后缀最大值 rmax[x]。

此时 maxx[x] = max(max(maxx[lc],maxx[rc]), rmax[lc]+lmax[rc]).

lmax 和 rmax 也需要同时更新。

为了更新 lmax 和 rmax, 我们还需要记录每个区间的区间和 sum.

lmax[x] = max(lmax[lc], sum[lc]+lmax[rc])

rmax[x] = max(rmax[rc], sum[rc]+rmax[lc])

sum[x] = sum[lc] + sum[rc]

例9extra spoj gss3 （线段树维护区间子段和）

一个长为 n 的序列 A，m次操作:

0 x y：把 A[x] 修改成y。(|y|<=10000)

1 x y: 输出 max{ A[i]+...A[j] },其中(x<=i<=j<=y)。即 [x, y]区间最大子段和(连续）。

| a[i] |<=10000, n,m <=50000。

sol

多了一个修改操作。

正常处理，修改操作合并（pushup)就行。

```
void change(int p,int v,int l,int r,int x){
    if(l==r){
        maxx[x]=lmax[x]=rmax[x]=sum[x]=v;
        return;
    }
    int mid=(l+r)>>1;
    if(p<=mid) change(p,v,l,mid,lc);
    else change(p,v,mid+1,r,rc);
    pushup(x);
}
```

例9extra2 spoj gss5 （线段树维护区间子段和）

一个长为 n 的序列 A， m次询问：

x1 y1 x2 y2: 输出 max{ A[i]+...A[j] },其中(x1<=i<=y1, x2<=j<=y2)。

保证 x1<=x2, y1<=y2

| a[i] |<=10000, n,m <=10000。

sol

区间查询限定了左右端点的范围。

那么我们需要分类讨论。

1、如果 [x1, y1] 和 [x2, y2] 没有交集，即 y1<x2, 显然答案为：

rmax([x1,y1]) + sum(y1+1, x2-1) + lmax([x2,y2])

2、如果有交集，即 y1>=x2:

此时区间分为三部分：[x1, x2-1]，[x2, y1], [y1+1,y2]

左端点有2种选择，右端点也有2种选择。

进一步讨论，变为三种情况。

maxx([x2,y1]) (左右端点都在2区间）

rmax([x1,x2-1]) + lmax([x2,y2])（左端点1 右端点2 3）

rmax([x1,y1]) + lmax([y1+1,y2])（左端点 1 2 右端点 3）

例10 hdu5649 dzy loves sorting（思路题）

一个数组 a 是一个长度为 n 的全排列。

现在有 m 次操作:

0 l r：对 a[l...r] 进行升序排列。

1 l r：对 a[l...r] 进行降序排列。

问经过 m 次操作，a[k] 为多少。

样例输入:

```
1
6 3
1 6 2 5 3 4
0 1 4
1 3 6
0 2 4
3
```

样例输出:

```
5
```

sol

每次对 [l, r] 执行一次排序，不现实。
用线段树标记完成对区间排序，也无法实现。

发现最后询问，且只询问一个位置上的数。
m次排序操作顺序不能乱，也无法离线。

sol

考虑二分答案 x。复杂度 O(mlogn)级别。

那么排列中，可以把大于等于 x 的数全设为 1，小于 x 的数全设为0。

我们发现，对 01 串进行排序，只需要把区间里的 0 移到最前面，1移到最后面即可。

对于要排序的区间 [l, r]:

先查询 [l, r] 的区间和 sum；然后将 [l, r-sum] 修改为 0，[r-sum+1, r] 修改为 1.

至此，我们就把排序变成了一次区间查询和区间覆盖操作。

sol

如何判断 x 比答案大还是小?

如果 a[k] = 0, 说明答案小于 x.

如果 a[k] = 1:

　　如果 a[k-1]=0,说明答案就是 x.

　　如果 a[k-1]=1,说明答案大于 x.

因此，线段树维护区间和和区间修改，最外层二分，时间复杂度为O(m*logn*logn)

核心代码

```
void pushup(int x){
    sum[x]=sum[lc]+sum[rc];
}
void pushdown(int l,int r,int x){
    if(tag[x]!=-1){
        int mid=(l+r)<<1;
        sum[lc]=(mid-l+1)*tag[x];
        sum[rc]=(r-mid)*tag[x];
        tag[lc]=tag[rc]=tag[x];
        tag[x]=-1;
    }
}
void build(int l,int r,int x,int v){
    tag[x]=-1;
    if(l==r){
        sum[x]=(a[l]>v);
        return;
    }
    int mid=(l+r)>>1;
    build(l,mid,lc);
    build(mid+1,r,rc);
    pushup(x);
}
```

```
int query(int a,int b,int l,int r,int x){
    if(a<=l&&r<=b) return sum[x];
    pushdown(l,r,x);
    int mid=(l+r)>>1;
    int ret=0;
    if(a<=mid) ret+=query(a,b,l,mid,lc);
    if(mid<b) ret+=query(a,b,mid+1,r,rc);
    return ret;
}
void change(int a,int b,int v,int l,int r,int x){
    if(l==r){
        sum[x]=v*(r-l+1);
        tag[x]=v;
        return;
    }
    pushdown(l,r,x);
    int mid=(l+r)>>1;
    if(a<=mid) change(a,b,v,l,mid,lc);
    if(mid<b) change(a,b,v,mid+1,r,rc);
    pushup(x);
}
```

核心代码

```
int main(){
    t=read();
    while(t--){
        n=read(),m=read();
        for(int i=1;i<=n;i++) a[i]=read();
        for(int i=1;i<=m;i++) op[i]=read(),l[i]=read(),r[i]=read();
        k=read();
        int L=1,R=n;
        while(L<=R){
            int mid=(L+R)>>1;
            build(1,n,1,mid);
            for(int i=1;i<=m;i++){
                int v=query(l[i],r[i],1,n,1);
                change(l[i],r[i],0,1,n,1);
                if(op[i]){
                    if(v>=1) change(l[i],l[i]+v-1,1,1,n,1);
                }else{
                    if(v>=1) change(r[i]-v+1,r[i],1,1,n,1);
                }
            }
            if(query(k,k,1,n,1)) L=mid+1;else R=mid-1;
        }
        cout<<L<<endl;
    }
    return 0;
}
```

P6025 线段树

【题目描述】

f[n] 表示 n 个节点的线段树使用数组的最大下标。

求 n 的取值的任意一个区间 [l, r] 的异或和，即

求 f[l] ⊕ f[l+1] ... ⊕ f[r] 的值。

【输入格式】

一行两个整数，表示 l,r.

【输出格式】

一行一个整数，表示结果。

P6025 线段树

【输入样例】

6 6

【输出样例】

13

【数据范围】

1<=l<=r<=1e15.

P6025 线段树

sol.

1、区间异或和 转 前缀异或和

$0\oplus a=a$，$a\oplus a=0$ =>

$f(l+1)\oplus f(l+2)...f(r)=f(1)\oplus...f(l-1)\oplus f(1)\oplus.f(l-1)..f(r)$

$=fsum(l-1)\oplus fsum(r)$

2、1<=t<=2^k-1时；f(2^k+2t)=f(2^k+2t+1)

n 从 2^k+2t 变成 2^k+2t+1 分裂的是左子树内的节点，此时右子树内最下面一层肯定有节点，所以答案不变。

3、利用倍增思想，在每个 2^k 到 2^(k+1)-1 区间，只有 f(2^k) 和 f(2^k+1) 对结果有影响。最后如果n是偶数且不为 2^k，则 f(n)也有影响。

4、n≠2^k或2^(k+1)，左子树全需要计算，因此 f(n)=2^dep+f(n/2).(其中dep为深度)

P3924 康娜的线段树

【题目描述】

如果每次在线段树区间加操作做完后，从根节点开始等概率的选择一个子节点进入，直到进入叶子结点为止，将一路经过的节点权值累加，最后能得到的期望值是多少？

每次会给你一个值 qwq，保证你求出的概率乘上 qwq 是一个整数。

【输入格式】

第一行整数 n,m,qwq 表示线段树维护的原序列的长度，询问次数，分母。

第二行 n 个数，表示原序列。

接下来 m 行，每行三个数 l,r,x 表示对区间[l,r] 加上 x

【输出格式】

共 m 行，表示期望的权值和乘上qwq结果。

P3924 康娜的线段树

【输入样例】

8 2 1

1 2 3 4 5 6 7 8

1 3 4

1 8 2

【输出样例】

90

120

【数据范围】

1<=n, m<=1e6, -1000<=ai, x<=1000.

P3924 康娜的线段树

sol.

1、本题计算期望权值和过程：

根节点层概率是1，所以期望的权值和是区间和乘以1，选中第二层某个区间的概率是1/2，所以期望权值和是所有节点的区间权值和乘以 1/2，第三层同理乘以 1/4 ... 直到叶子节点为止。

2、整数运算：

相当于是每个节点所代表的区间和除以 2^(dep-1)，其中dep为该节点深度。

为了方便运算，我们可以在建树过程中预处理出最大深度 md，然后各节点乘以2^(md-dep)，最后求答案时共同除以 2^(md-1)即可。

于是求解就是从根节点到叶节点经过的所有点权值分别乘以 2^(md-dep)。

P3924 康娜的线段树

sol.

3、考虑区间加操作对结果的影响：

我们发现如果某个点加了一个 x，计算时还是需要从根节点走到叶节点，过程中分别乘以所经过节点的 2^(md-dep).

因此相当于答案就会增加 x $*\sum_{i=1}^{dep}\frac{1}{2\text{\textasciicircum}(i-1)}$ ，通过等比数列求和公式可化简为 x $*\ \frac{2^{dep}-1}{2^{dep-1}}$ ，根据第二步整数运算，所以我们只需计算 x $*\ (2^{dep}-1)\ *\ 2^{\mathrm{md}-dep}$ 即可，最后统一除 2^(md-1).

对于区间 [l,r] 连续多个点，每个点预处理出对应的 $(2^{dep}-1)\ *\ 2^{\mathrm{md}-dep}$ ，然后乘以 x 即为对该区间加 x 对结果的影响。

一个优化是我们可以在建树的时候同时记录每个叶节点的深度 dep,然后利用前缀和的思想预处理出 s[i] 表示从 1 到 i 的 $(2^{dep}-1)\ *\ 2^{\mathrm{md}-dep}$值的和，那区间 [l,r] 即为 s[r]-s[l-1].

最后记得乘以 qwq ,再除以 2^(md-1)即为每次操作的答案。（这个运算可能先要约分）。