线段树

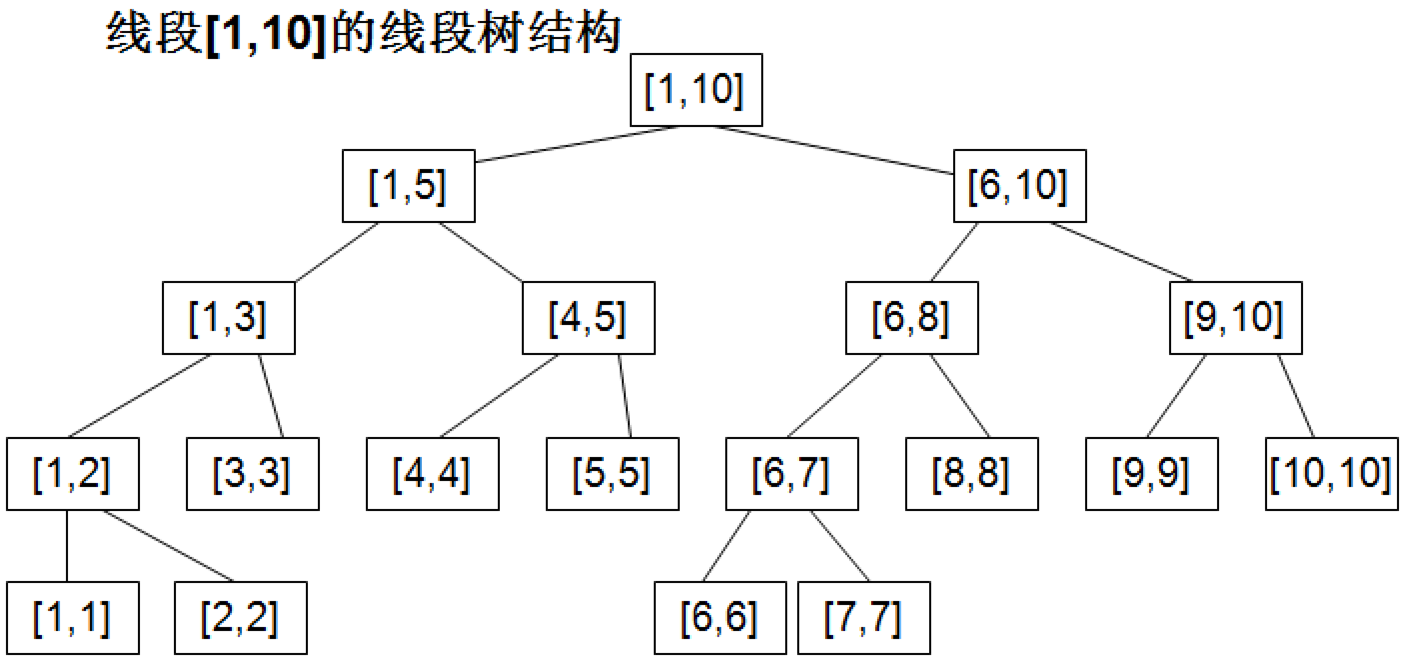
石室中学：梁德伟

线段树

- 用途：查询区间最值，查询区间和，而且能动态更新。
- 树状数组，st表能干的他都能干！

线段树的构造

- 分治思想
- 维护区间和
  - 单点操作
  - 区间操作



对于一颗二叉树

大概有$1+2^1+2^2+2^3+2^4+...+Lenth \approx 4 * Lenth$

为了保险我们数组开到Lenth的4倍大

```
#define lc (p<<1)
#define rc (p<<1|1)
struct Node{
    int l,r,sum;
};
Node T[4*N];
void pushup(int p){
    T[p].sum=T[lc].sum+T[rc].sum;
}
void build(int p,int l,int r){//建树
    T[p].l=l;T[p].r=r;
    if(l==r){
        T[p].sum=0;return;
    }
    int mid=(l+r)>>1;
    build(lc,l,mid);
    build(rc,mid+1,r);
    pushup(p);
}
```

```
void update(int p,int k,int v){//单点修改
    if(T[p].l==T[p].r){
        T[p].sum+=v;
        return;
    }
    int mid=(T[p].l+T[p].r)>>1;
    if(k<=mid)
        update(lc,k,v);
    else
        update(rc,k,v);
    pushup(p);
}
int query(int p,int ql,int qr){
    if(ql<=T[p].l&&qr>=T[p].r)
        return T[p].sum;
    int mid=T[p].l+T[p].r>>1;
    int ans=0;
    if(ql<=mid)  ans+= query(lc,ql,qr);
    if(qr>mid)   ans+= query(rc,ql,qr);
    return ans;
}
```

例：A2120

单点修改，区间查询模板题

就这么简单？

- 那么恶意加大数据范围怎么样？

给你N个数，有两种操作：

1：给区间[a,b]的所有数增加X

2：询问区间[a,b]的数的和。

1<=n<=200000

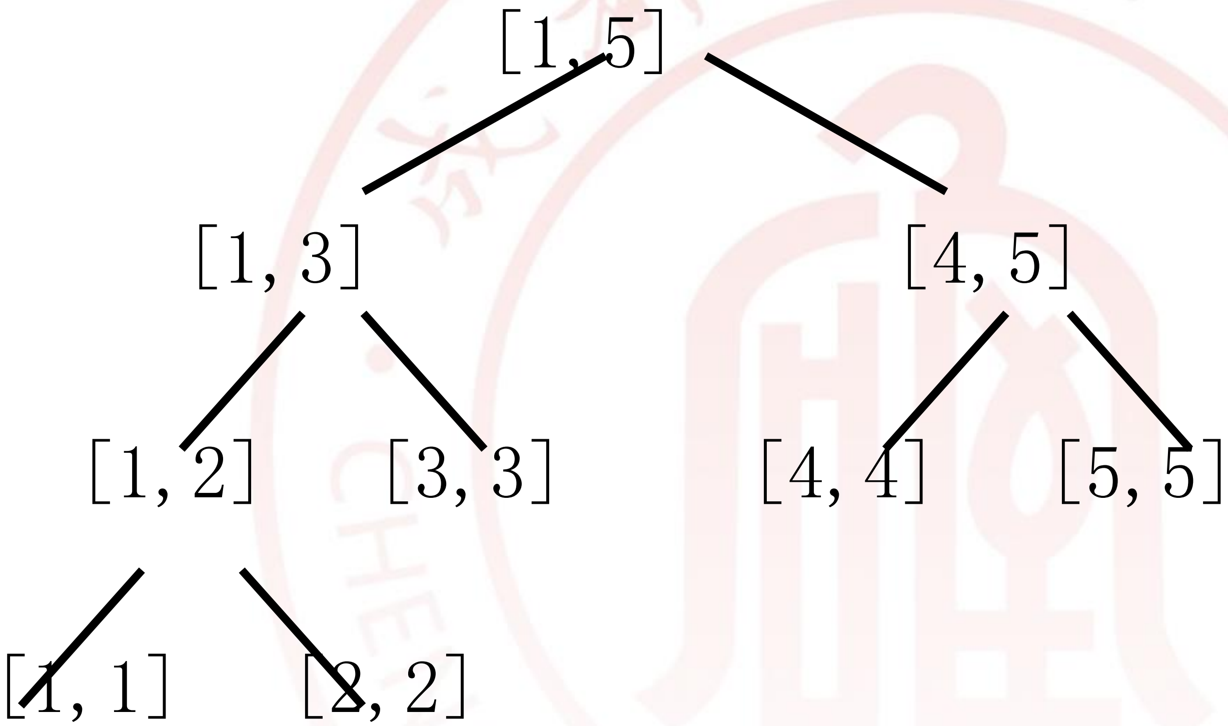
1<=q<=200000

如果我们暴力更新（更高级的是我们进行线段树上的暴力），那么我们将进行200000*200000*4次更新......

延迟标记

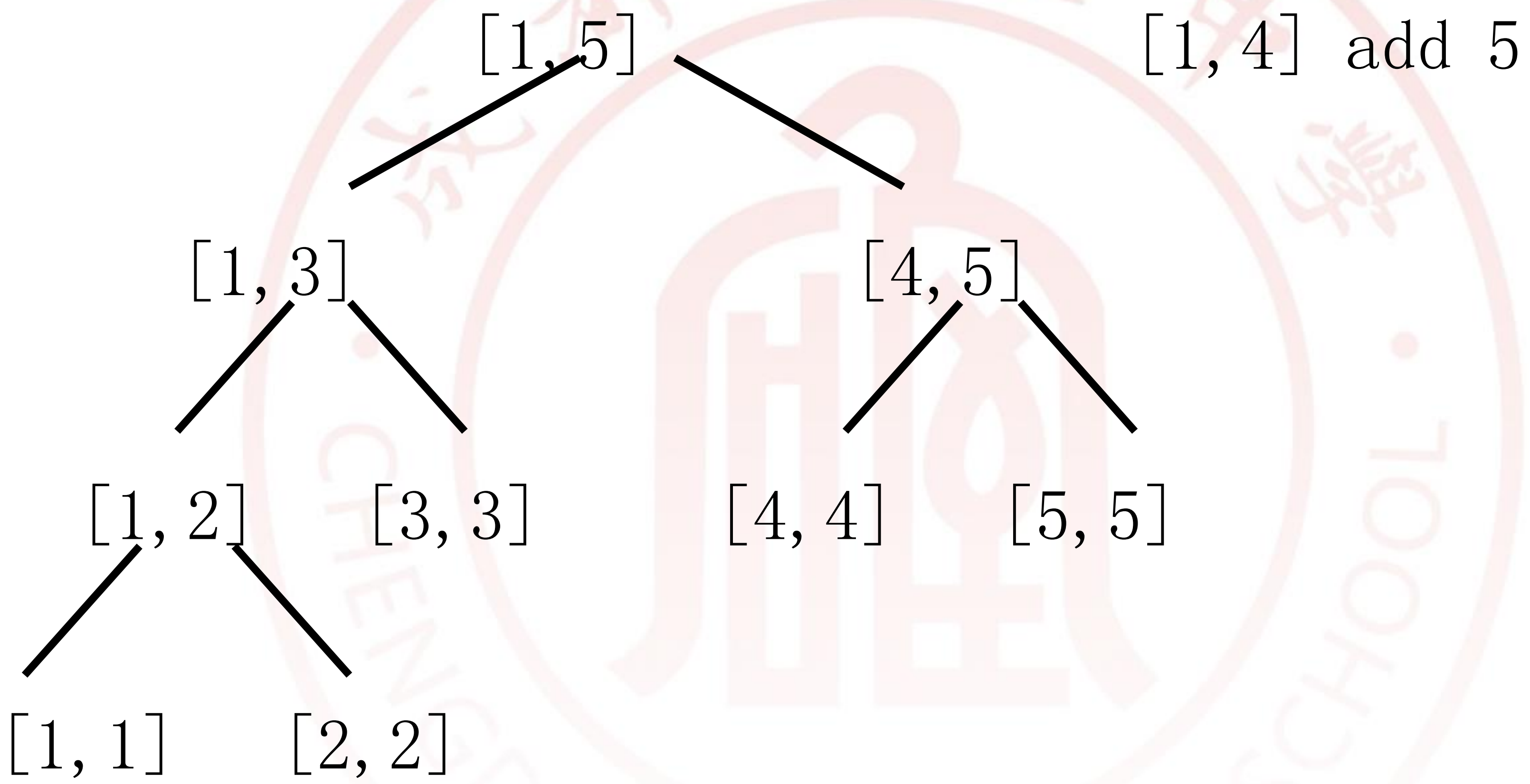
- 将区间修改的操作，在线段树上对应的区间节点做一个标记。当需要访问或修改该节点下面子树时才将标记下放。这样的思想称为延迟标记思想

A Simple Problem with Integers

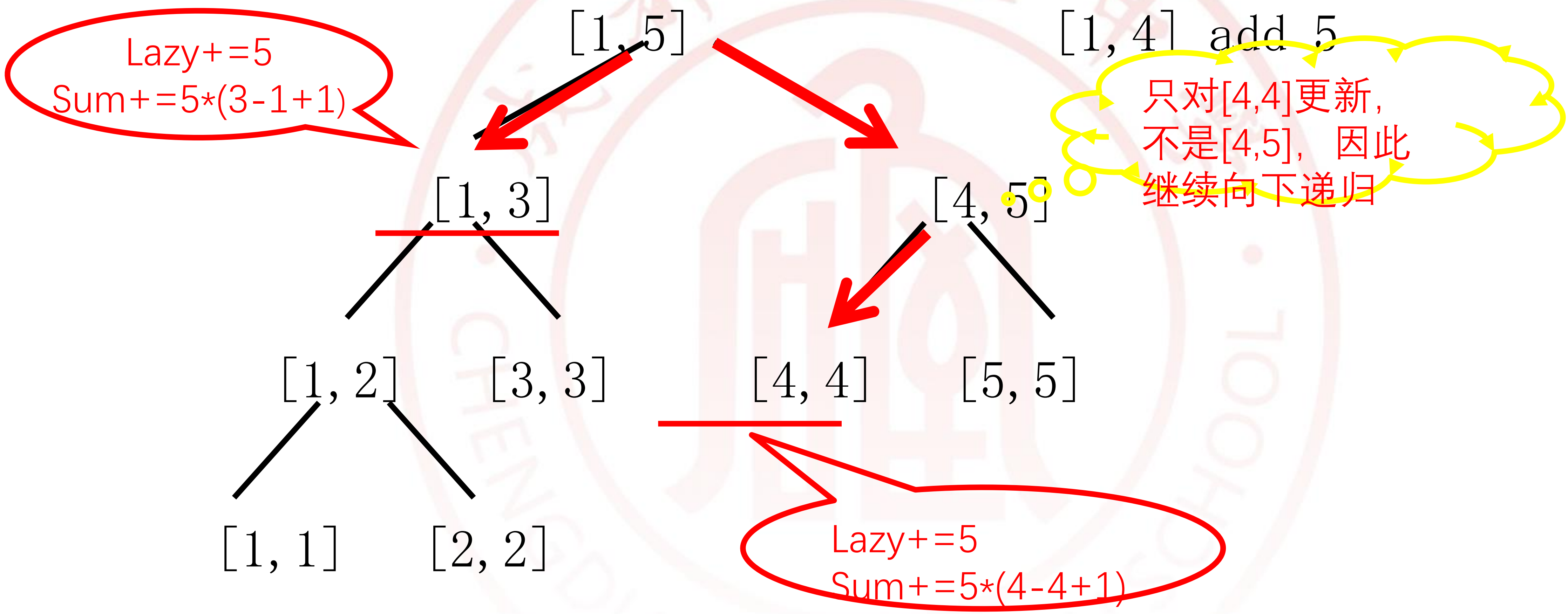



```
struct Segtree {
    int l, r;
    Type sum;
    Type lazy;
} tr[N * 4];
```

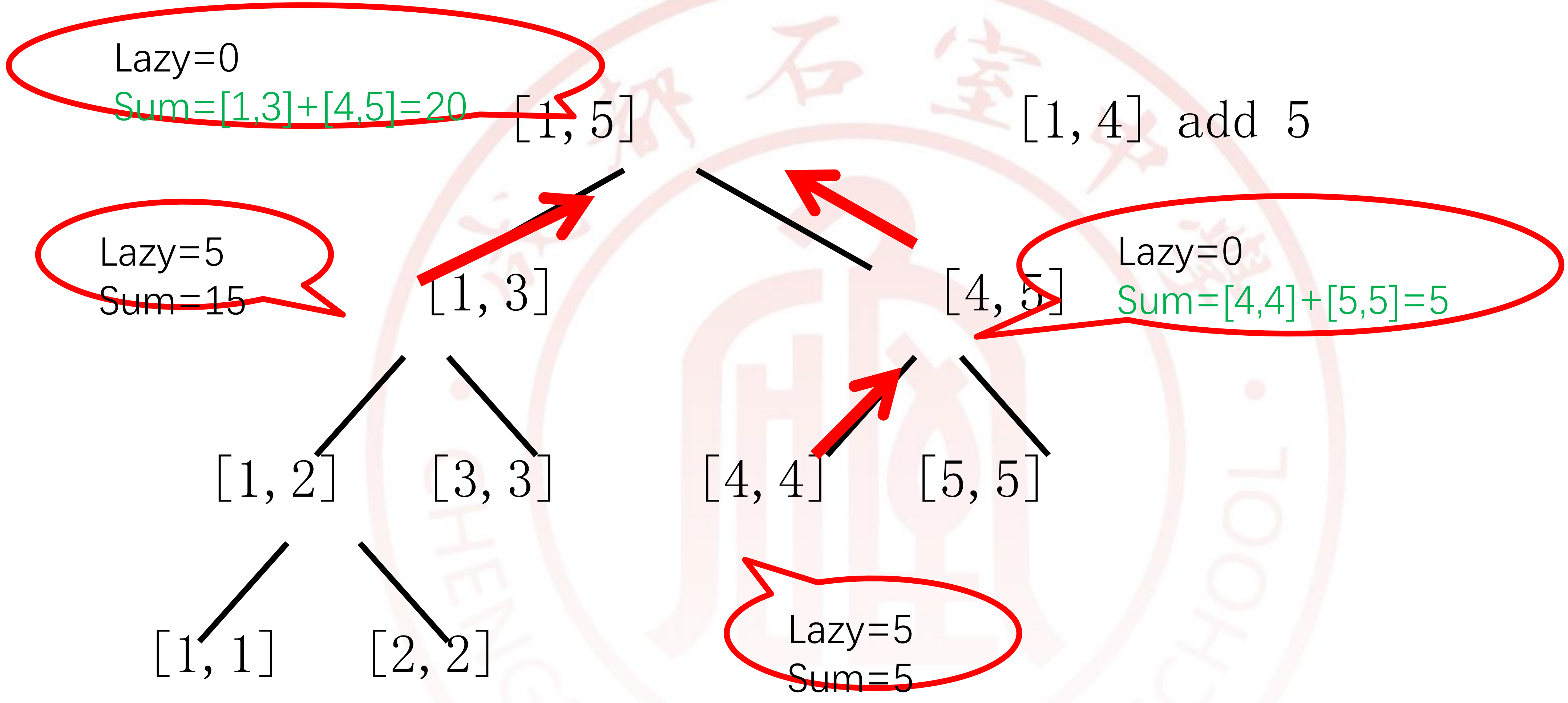
A Simple Problem with Integers



A Simple Problem with Integers

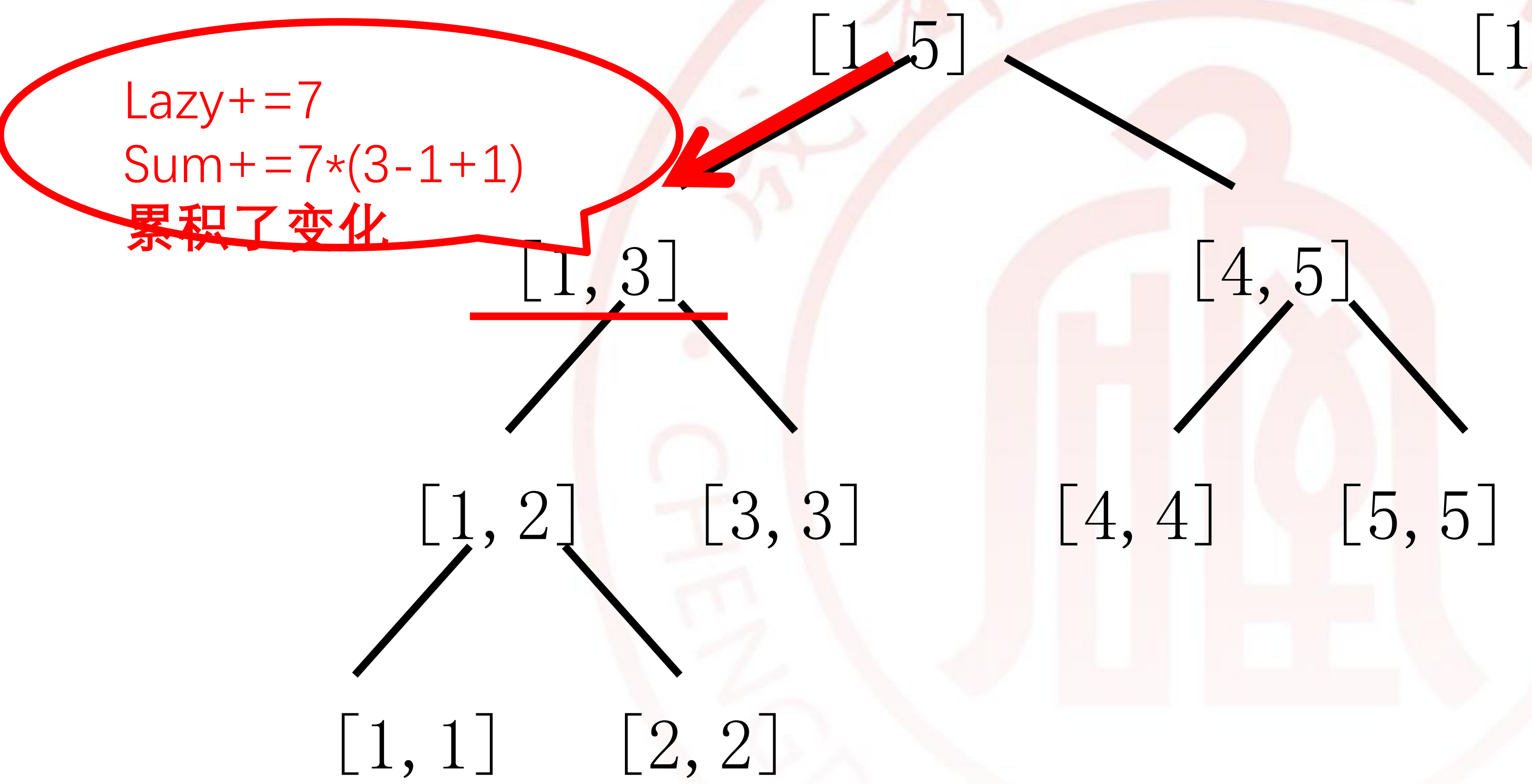


A Simple Problem with Integers

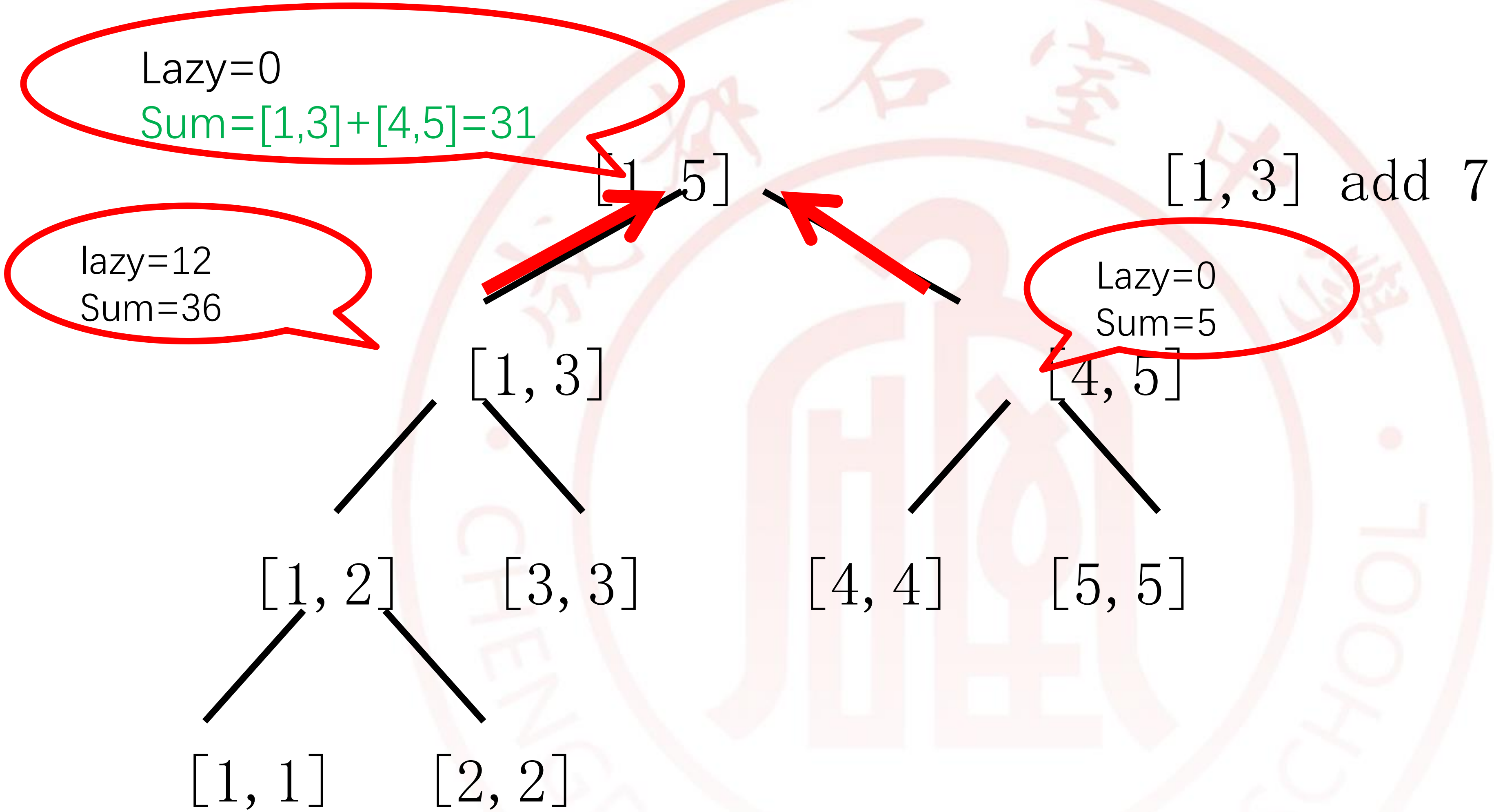


A Simple Problem with Integers

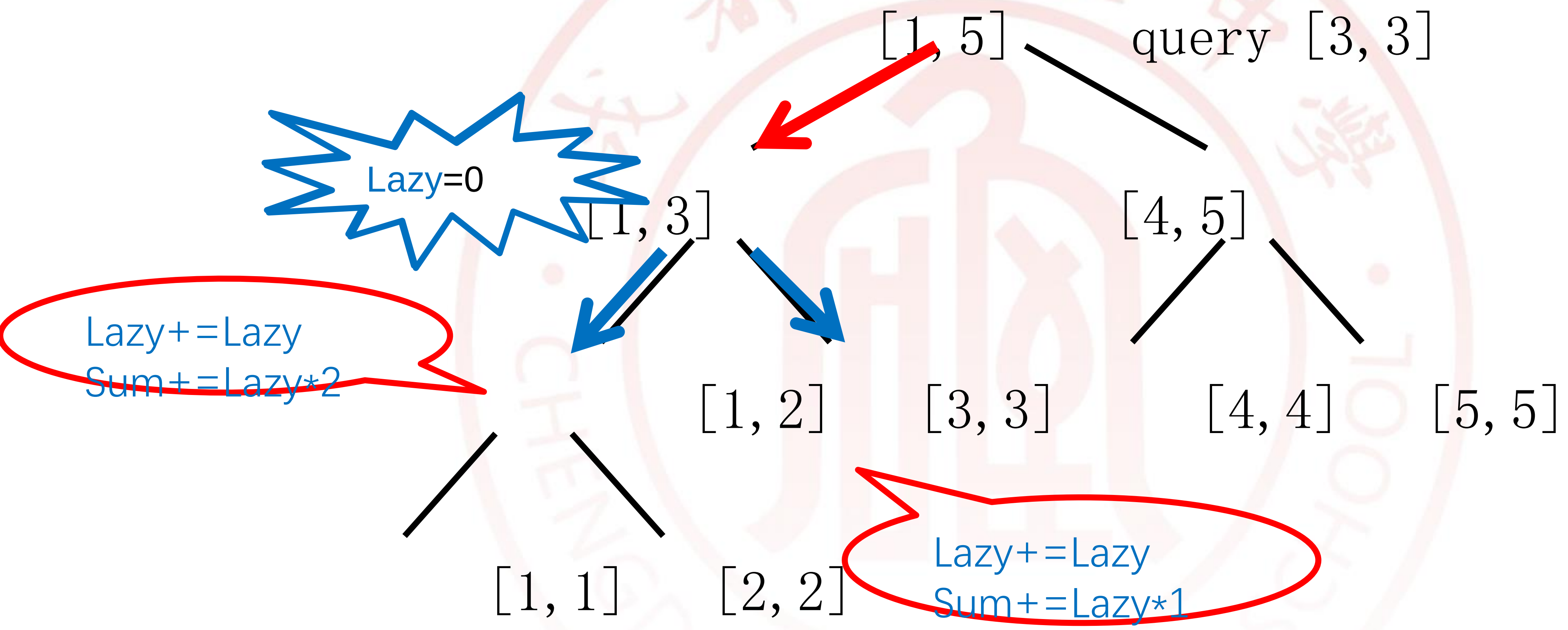
[1, 3] add 7



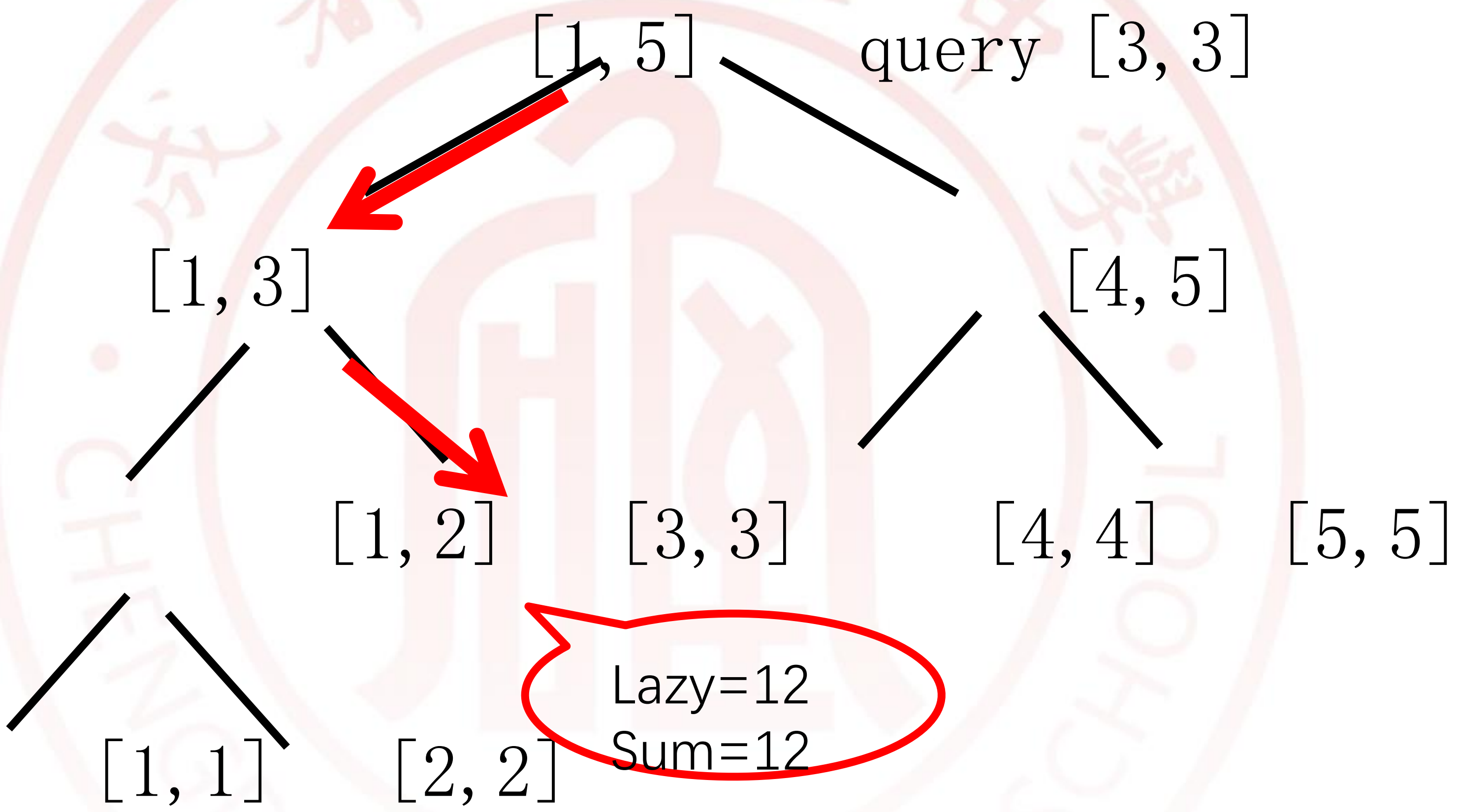
A Simple Problem with Integers



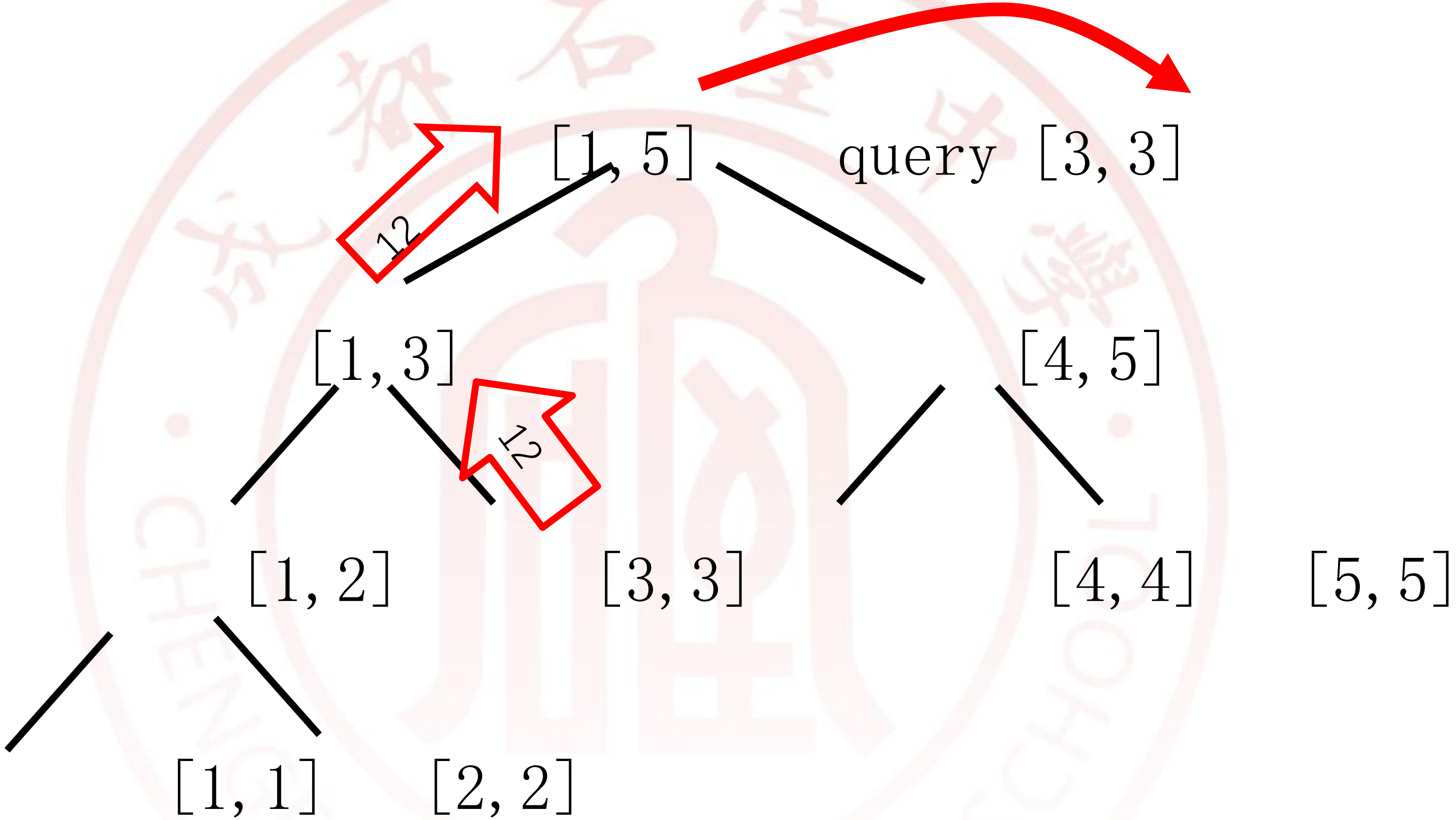
A Simple Problem with Integers



A Simple Problem with Integers



A Simple Problem with Integers



一般过程

- 从上面的例子中，我们可以总结出，线段树上维护的信息通常有两类：

- （1）诸如sum、max、min 之类的信息，表示了一个结点的性质，具有一定的合并性质，可以在维护的时候自下向上(push_up)地合并计算。

- （2）诸如lazy之类的信息， 包含了整个子树的要进行处理的信息，可以在需要进一步处理时，将这种信息自上而下(push_down)传递。

一般过程

- 于是，我们可以总结出一个线段树上更新或者查询区间的一般过程：
  - 1. 对于当前待处理的区间[l，r]，检查是否与当前节点管辖区间相同，若相同，则进行相应处理并返回
  - 2. 下放延迟标记
  - 3. 根据待处理区间与左右儿子区间的关系进行递归处理
  - 4. 通过左右儿子合并更新该节点维护的信息

```
#define LL long long
#define lc (p<<1)
#define rc (p<<1|1)
struct Node{
    int l,r,lazy;
    LL sum;
};
Node T[100010*4];
void pushnow(LL p,LL v){
    T[p].sum+=(T[p].r-T[p].l+1)*v;
    T[p].lazy+=v;
}
void pushdown(LL p){
    if(T[p].lazy){
        pushnow(lc,T[p].lazy);
        pushnow(rc,T[p].lazy);
        T[p].lazy=0;
    }
}
void pushup(LL p){
    T[p].sum=T[lc].sum+T[rc].sum;
}
void build(LL p,LL l,LL r){//建树
    T[p].l=l;T[p].r=r;
    if(l==r){
        T[p].sum=a[l];T[p].lazy=0;
        return;
    }
    LL mid=(l+r)>>1;
    build(lc,l,mid);
    build(rc,mid+1,r);
    pushup(p);
```

```
void update(LL p,LL ql,LL qr,LL v)//向上维护
{
        if(ql<=T[p].l&&T[p].r<=qr){
                pushnow(p,v);
                return;
        }
        LL mid=(T[p].l+T[p].r)>>1;
        pushdown(p);
        if(ql<=mid)
update(lc,ql,qr,v);
        if(qr>mid)
                update(rc,ql,qr,v);
        pushup(p);
}
LL query(LL p,LL ql,LL qr)
{
        if(ql<=T[p].l&&qr>=T[p].r)
                return T[p].sum;
        LL mid=T[p].l+T[p].r>>1;
        pushdown(p);
        LL ans=0;
        if(ql<=mid)
                ans+=query(lc,ql,qr);
        if(qr>mid)
                ans+= query(rc,ql,qr);
        pushup(p);
        return ans;
}
```

懒标记的局限性

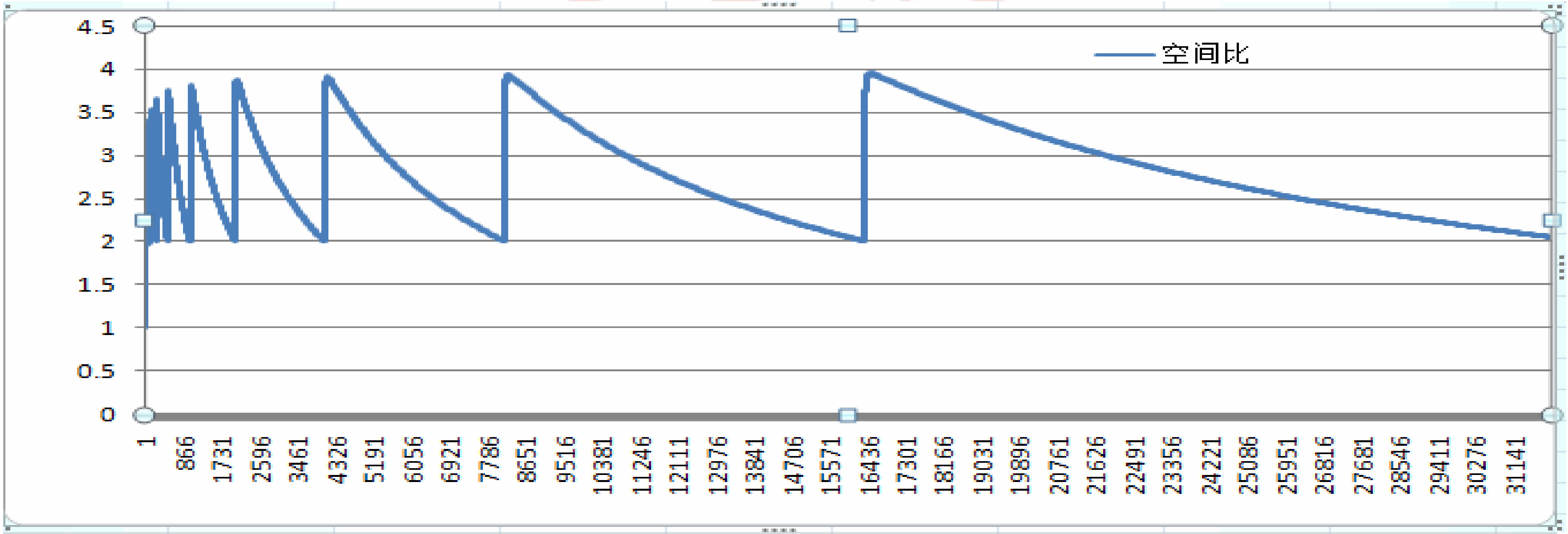
- 区间开根号，询问区间和
- 一些非典型的线段树方法。

线段树——空间复杂度

- 设长度为N的数组在线段树中，编号最靠右的节点编号为F(N)
  - 若$N=2^n$，$F(N)=2^{(n+1)}$
  - 若$N=2^{(n+1)}$，$F(N)=2^{(n+2)}$

- 因而对于$2^n<=N<=2^{(n+1)}$，有
- $2^{(n+1)}<=F(N)<=2^{(n+2)}$
- $F(N)<=4*N$

线段树——空间复杂度（II）



（图片转自http://comzyh.tk/blog/archives/479/）

线段树空间应开为原数组长度的4倍

线段树——小结

- 1、线段树可以做很多很多与区间有关的事情……
- 2、空间复杂度~O(N*4)，每次更新和查询操作的复杂度都是O(logN)。
- 3、在更新和查询区间[l,r]的时候，为了保证复杂度是严格的O(logN)必须在达到被[l,r]覆盖的区间的结点时就立即返回。而为了保证这样做的正确性，需要在这两个过程中做一些相关的“懒”操作。
- “懒操作”在更新区间的有关问题上至关重要。

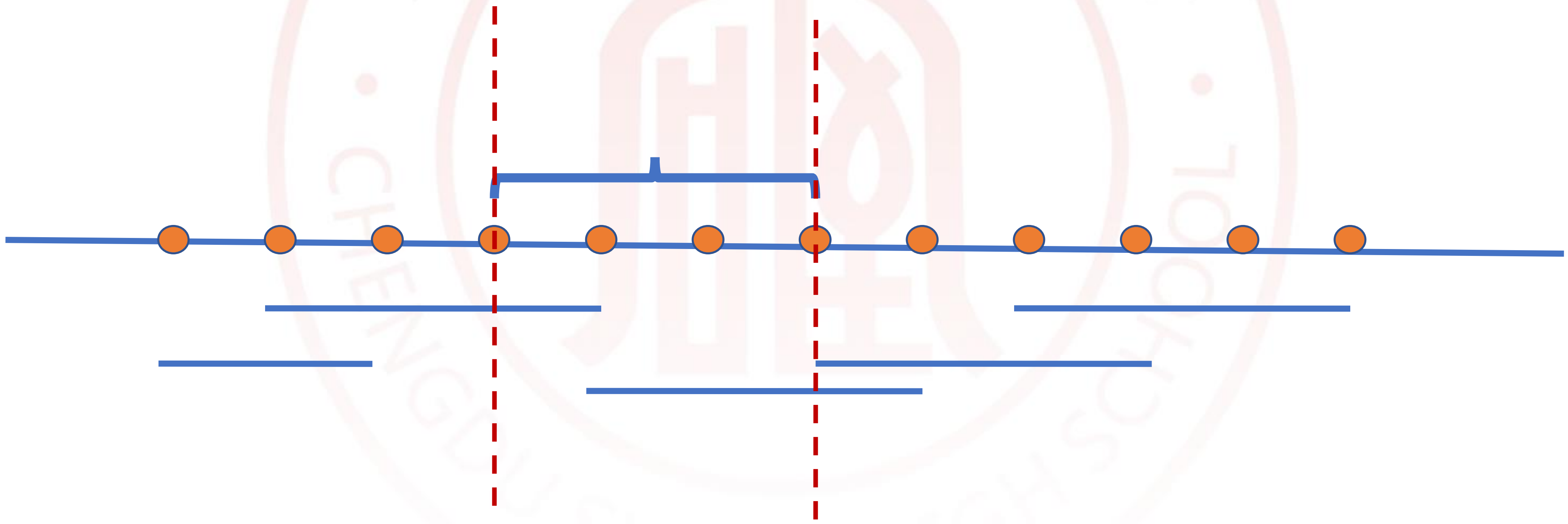
区间乘，区间查询

- 增加乘法的lazy-mul
- 注意，乘法的lazy-mul，是区间和*lazy-mul。
- 同时加法lazy-add*lazy-mul

```
inline void pushdown(LL p){
    if(T[p].mul!=1){
        T[lc].sum=T[lc].sum*T[p].mul%mod;
        T[rc].sum=T[rc].sum*T[p].mul%mod;
        T[lc].mul=T[lc].mul*T[p].mul%mod;
        T[rc].mul=T[rc].mul*T[p].mul%mod;
        T[lc].add=T[lc].add*T[p].mul%mod;
        T[rc].add=T[rc].add*T[p].mul%mod;
        T[p].mul=1;
    }
    if(T[p].add){
        T[lc].sum+=T[p].add*len(lc)%mod;
        T[lc].add+=T[p].add;
        T[rc].sum+=T[p].add*len(rc)%mod;
        T[rc].add+=T[p].add;
        T[p].add=0;
    }
}
```

例1：P2555贪婪大陆

- 【题意】插入若干条线段，问任意区间上有多少条线段。



分析

- 要统计每个区间的线段树，即对于每个节点需要维护有多少线段。但是思考后发现，区间线段数很难维护。

- 正难则反

分析

- 我们发现，求区间[l,r]的线段树，只需要求出不在[l,r]的线段数T。然后用总线段树减去T即是答案。
- 如何维护多少线段不在区间内呢？

分析

- 我们发现，求区间[l,r]的线段树，只需要求出不在[l,r]的线段数T。然后用总线段树减去T即是答案。
- 如何维护多少线段不在区间内呢?
- 一个结论：
  - 对于区间[a,b]，如果一些线段右端点在[1,a-1]，则该线段与[a,b]一点不重合，同理，如果线段的左端点落在[b+1,n]，则与[a,b]不重合

设计算法

- 2颗线段树维护每个点的右端点数、左端点数。
- 当然，实际操作只需要1颗线段树，每个节点维护2个值。

```
int main(){
    scanf("%d%d",&n,&q);
    build(1,1,n);
    while(q--){
        int t,x,y;
        scanf("%d%d%d",&t,&x,&y);
        if(t==1){
            last++;
            update(1,y,y,1,1);//右端点
            update(1,x,x,1,0);//
        }
        else{
            printf("%d\n",last-query(1,1,x-1,1)-query(1,y+1,n,0));
        }
    }
    return 0;
}
```

```cpp
#define N 100100
#define lc (p<<1)
#define rc (p<<1|1)
struct Node{
	int l,r,lazy[2];
	int sum[2];
};
Node T[4*N];
int n,q,last=0;
void pushnow(int p,int k,int d){//d=0表示修改左端点信息
	T[p].sum[d]+=(T[p].r-T[p].l+1)*k;
	T[p].lazy[d]+=k;
}
void pushup(int p){
	T[p].sum[0]=T[lc].sum[0]+T[rc].sum[0];
	T[p].sum[1]=T[lc].sum[1]+T[rc].sum[1];
}
void pushdown(int p){
	for(int i=0;i<2;i++)
		if(T[p].lazy[i]){
			pushnow(lc,T[p].lazy[i],i);
			pushnow(rc,T[p].lazy[i],i);
			T[p].lazy[i]=0;
		}
}
```

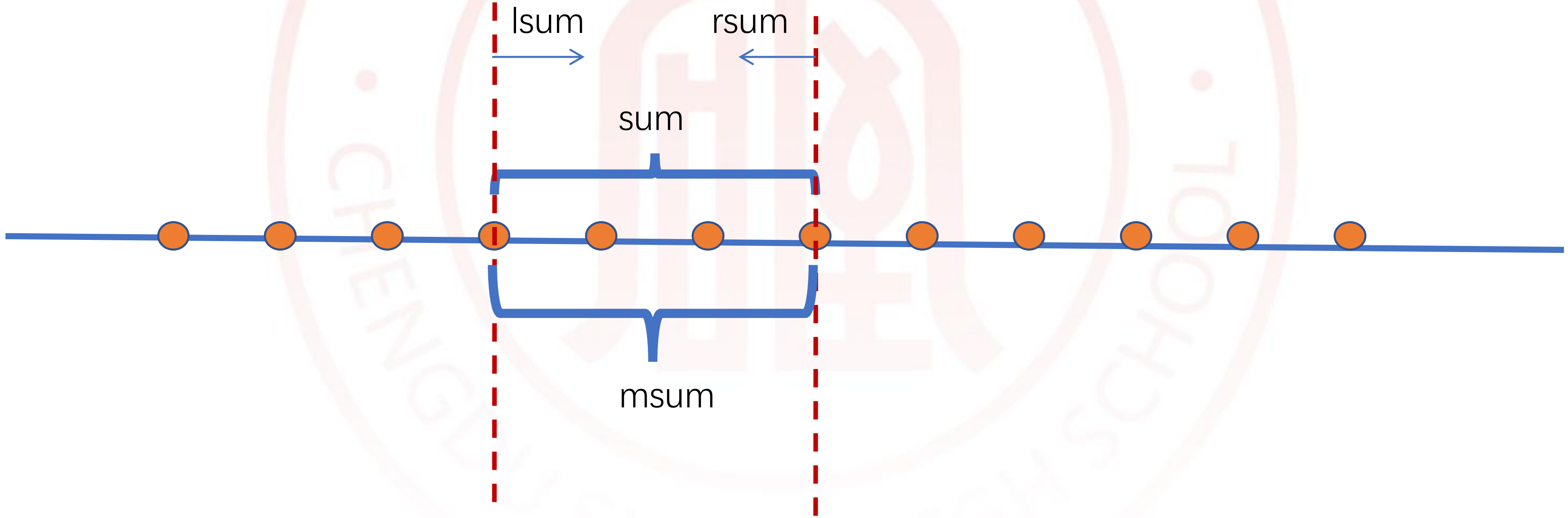
```
void update(int p,int ql,int qr,int x,int d){
    if(qr<T[p].l||ql>T[p].r)return ;
    if(ql<=T[p].l&&T[p].r<=qr){
        pushnow(p,x,d);
    }
    else{
        int mid=T[p].l+T[p].r>>1;
        pushdown(p);
        if(ql<=mid)update(lc,ql,qr,x,d);
        if(qr>mid)update(rc,ql,qr,x,d);
        pushup(p);
    }
}
int query(int p,int ql,int qr,int d){
    if(qr<T[p].l||ql>T[p].r)return 0;
    if(ql<=T[p].l&&T[p].r<=qr){
        return T[p].sum[d];
    }else{
        int ans=0;
        int mid=T[p].l+T[p].r>>1;
        pushdown(p);
        if(ql<=mid)ans+=query(lc,ql,qr,d);
        if(qr>mid)ans+=query(rc,ql,qr,d);
        pushup(p);
        return ans;
    }
```

例2：　P2560程序设计竞赛

- 【题意】单点修改，查询区间内连续序列最大值

思路

- 维护4个值：



例3：P2614 Hotel

- 题意：初始一个为0的序列，2个操作
- 操作1：查询最靠左的连续长度为D1的连续零的区域并置为1
- 操作2：将区间[l,r]置零

例3： P2614 Hotel

- 成段更新,寻找空间(经典类型,求一块满足条件的最左边的空间)

- 线段树维护三个值，区间内最大的连续空位，从左边最大延伸的长度，从右边最大延伸的长度。 更新时，有人住让这些数组都为0，表示0个空位，没人住，就是区间长度，表示区间内的空位就是长度，都是空位。

```cpp
#define lc (p<<1)
#define rc (p<<1|1)
int sum[maxn<<2],len[maxn<<2],lsum[maxn<<2],rsum[maxn<<2],lazy[maxn<<2];
int n,m;
inline void pushup(int p){
    if(sum[lc] == len[lc])lsum[p]=sum[lc]+lsum[rc];
    else lsum[p]=lsum[lc];
    if(sum[rc]==len[rc])rsum[p]=sum[rc]+rsum[lc];
    else rsum[p]=rsum[rc];
    sum[p]=max(rsum[lc]+lsum[rc],max(sum[lc],sum[rc]));
}
inline void pushdown(int p){
    if(!lazy[p])return ;
    if(lazy[p]==1){//住房
        lazy[lc]=lazy[rc]=1;
        sum[lc]=sum[rc]=lsum[lc]=lsum[rc]=rsum[lc]=rsum[rc]=0;
    }
    if(lazy[p]==2){
        lazy[lc]=lazy[rc]=2;
        sum[lc]=lsum[lc]=rsum[lc]=len[lc];
        sum[rc]=lsum[rc]=rsum[rc]=len[rc];
    }
    lazy[p]=0;
}
```

```cpp
void build(int p,int l,int r){
    sum[p]=len[p]=lsum[p]=rsum[p]=r-l+1;lazy[p]=0;
    if(l==r)return;
    int mid=(l+r)>>1;
    build(lc,l,mid);build(rc,mid+1,r);
}
void update(int p,int l,int r,int ql,int qr,int d){
    if(ql<=l&&r<=qr){
        lazy[p]=d;
        if(d==1)sum[p]= lsum[p]=rsum[p]=0;//住房
        else sum[p]=lsum[p]=rsum[p]=len[p];
        return;
    }
    pushdown(p);
    int mid=(l+r)>>1;
    if(ql<=mid)update(lc,l,mid,ql,qr,d);
    if(qr>mid)update(rc,mid+1,r,ql,qr,d);
    pushup(p);
}
int query(int p,int l,int r,int x){
    pushdown(p);
    int mid=(l+r)>>1;
    if(sum[lc]>=x)return query(lc,l,mid,x);//左区间够，左边找
    if(rsum[lc]+lsum[rc]>=x)return mid-rsum[lc]+1 ;//中间够端点在中间
    return query(rc,mid+1,r,x); //一定在右边
}
```

```cpp
signed main(){
    n=in;m=in;
    build(1,1,n);
    while(m--){
        int op,x,y,ans;
        op=in;
        if(op==1){
            x=in;
            if(sum[1]<x){
                puts("0");continue;
            }
            ans=query(1,1,n,x);
            printf("%d\n",ans);
            update(1,1,n,ans,ans+x-1,1);//申请住宿

        }else{
            x=in;y=in;
            update(1,1,n,x,x+y-1,2);//退房
        }
    }
    return 0;
}
```

小结

1、线段树用于序列区间查询值的维护

2、区间查询，主要考虑答案是否满足区间加法原则，如果不能直接满足，则需要通过一些中间变量来维护答案

权值线段树

- 普通线段树，相当于数组下标为基础建立的线段树。权值线段树，顾名思义就是以数值大小建立线段树（类似计数排序与普通排序）

- 由于是以数值大小建立线段树，数值过大时需要先离散化

T1：POJ2528

- 题目大意：给你一个无限长的板子，然后依次往上面贴n张等高的海报，问你最后能看到多少张海报。

POJ2528

- 题目大意：给你一个无限长的板子，然后依次往上面贴n张等高的海报，问你最后能看到多少张海报。
- 求解：对区间离散化，然后处理

离散化

- 举个例子：
  - 原数组ax [-1, 120, 13, 45, 12, 12]
  - 排序去重后得到[-1, 12, 13, 45, 120]
  - 映射完后得到新的ax数组 [1,5,3,4,2,2]
- 一种比较简单的写法：
  - 将所有操作到的数用一个数组存起来，然后排序，去重，该数在数组中的下标就是映射后的新的编号。

离散套路

- Sort+unique
- sort(a , a+tot);//tot是数组长度 int m=unique(a,a+tot)-a;

回到问题

- 对数值离散后，问题可以等价于：

长度为L的线段，每次对一个区间染色，最后能看见几种颜色，L<=1e5

有两个思路：

方法1：

对修改操作倒序操作，每次修改一个区间，如果修改的区间全部都被染过颜色，则忽略当前颜色，否则颜色数量+1

方法2

- 离散后每个修改做区间覆盖修改

- 最后统计一共多少颜色即可

- 不过对于本题离散后有一个细节需要注意
- 例子一:1-10 1-4 5-10
- 例子二:1-10 1-4 6-10
- 普通离散化后都变成了[1,4][1,2][3,4]
- 线段2覆盖了[1,2],线段3覆盖了[3,4],那么线段1是否被完全覆盖掉了呢?
- 例子一是完全被覆盖掉了,而例子二没有被覆盖

- 解决的办法则是对于距离大于1的两相邻点，中间再插入一个点

```cpp
#define lc (o<<1)
#define rc (o<<1|1)
int ll[N],rr[N],vis[N],lazy[N],a[N],ans;
int t,n,m,tot;
void pushdown(int o){
    if(lazy[o]==0)return;
    lazy[lc]=lazy[rc]=lazy[o];
    lazy[o]=0;
}
void update(int o,int l,int r,int ql,int qr,int x){
    if(ql<=l && r<=qr){
        lazy[o]=x;return;
    }
    pushdown(o);
    int mid=(l+r)>>1;
    if(qr<=mid)update(lc,l,mid,ql,qr,x);
    else if(ql>mid)update(rc,mid+1,r,ql,qr,x);
    else update(lc,l,mid,ql,mid,x),update(rc,mid+1,r,mid+1,qr,x);
}
void query(int o,int l,int r){
    if(lazy[o]>0){
        if(vis[lazy[o]] == 0)   ans++,vis[lazy[o]]=1;
        return;
    }
    if(l==r)return;
    int mid=(l+r)>>1;
    query(lc,l,mid);query(rc,mid+1,r);
}
```

```cpp
int main(){
    scanf("%d",&t);
    while(t--){
        memset(lazy,0,sizeof(lazy));
        memset(vis,0,sizeof(vis));
        scanf("%d",&n);
        for(int i=1;i<=n;i++){
            scanf("%d%d",ll+i,rr+i);
            a[i*2-1]=ll[i];
            a[i*2]=rr[i];
        }
        sort(a+1,a+2*n+1);
        m=unique(a+1,a+2*n+1)-a-1;      //
        tot=m;
        for(int i=1;i<m;i++){
            if(a[i+1]-a[i]>1) a[++tot]=a[i]+1;
        }
        sort(a+1,a+tot+1);
        for(int i=1;i<=n;i++){
            int x=lower_bound(a+1,a+tot+1,ll[i])-a;
            int y=lower_bound(a+1,a+tot+1,rr[i])-a;
            update(1,1,tot,x,y,i);
        }
        ans=0;
        query(1,1,tot);
        printf("%d\n",ans);
    }
    return 0;
}
```

T2： P2616

- 题意：给出m个数wi，表示每天电影的权值，再给出n个数，即每天播放的电影。
- 找出一个区间求最大权值和（不算重复）

P2616

- 题意：给出m个数wi，表示每天电影的权值，再给出n个数，即每天播放的电影。
- 找出一个区间求最大权值和（不算重复）
- 本题抓住每部电影只能看一次。在保证一部电影只看一次的情况下肯定看更多电影最好，因此，考虑每个点作为答案区间的左端点，则答案区间最大可能为[i,next[i]-1](next[i]表示这部电影下一个播放时间点，反过来说，对于电影i只对区间[i,next[i]-1]有贡献。
- 采取线段树维护最大值，每个点对特殊的区域有贡献（有贡献即将该区域全部值+w[i]）。枚举每个点为左端点，动态维护线段树的值。如何维护呢？当假设点i有贡献，则相当于区间[i,next[i]-1]全部+w[i]，考虑i+1时，则需要将i的贡献去掉，即[i,next[i]-1]全部-w[i]，[next[i],next[next[i]-1]+w

T4：市场loj6029

- 题意：修改：区间加、区间除
- 询问：区间求最小，区间和

做法

- 主要考虑除法，区间做除法可以考虑为区间做减法，你们对于[l,r]做除法肯定不能对这个区间做一次减法，因为要减去 的数不一样，但是可以考虑到某子区间需要减去的数字可以是一样的，就可以用区间更新的方法整段更新即可。
- 而如何判断一个区间减去数一样呢，主要看最大值和最小值减去多少，如果是一样的，那么就说明这个区间减去数是一样的
- 其实题目让求最小值已经是做了提示

T5：P2617区间取模，区间求和

T5：　P2617区间取模，区间求和

- 做法：本题关键在证明对于区间取模，每个点只会被修改LOG次
- 类似的还有区间开根号,区间取Phi（hdu5634）
- 具体做法是设一个最大值，如果区间最大值<mod就不修改，否则暴力修改

练习表

- 2212
- 2126
- 2325
- 2577

- 博客参考 https://blog.csdn.net/dreaming__ldx/article/details/81261996