

# 线段树 part2

SegmentTree 重庆南开中学信竞基础课程

# 别慌，先复习一下！

```
void MakeTree(int p,int x,int y)    //建立表示区间[x,y]的线段树，p为当前节点编号
{
    Tree[p].a= x;   Tree[p].b= y;    //为新节点表示的左右区间赋值
    if (x < y)
    {
        MakeTree(p*2, x , (x+y)/2 );    //递归建左子树
        MakeTree(p*2+1, (x+y)/2+1 , y);    //递归建右子树
    }
}
```

## //更快的建树代码

```
void MakeTree(int p,int x,int y)    //建立表示区间[x,y]的线段树，p为当前节点编号
{
    Tree[p].a= x;   Tree[p].b= y;    //为新节点表示的左右区间赋值
    if (x >= y) return;

    int mid=(x+y)>>1;
    MakeTree(p<<1, x , mid );    //p<<1 等价于 p*2
    MakeTree(p<<1|1, mid+1 , y);    //p<<1|1 等价于p*2+1
}
```

# 线段树的Lazy标记

## 例1：数列操作2 NK0J2297

**描述：**给出一列数  $\{A_i\}$  ( $1 \leq i \leq n$ )，总共有  $m$  次操作，操作分两种：

1. ADD  $x \ y \ z$  将  $x$  到  $y$  区间的所有数字加上  $z$
2. ASK  $x \ y$  将  $x$  到  $y$  区间的最大一个数字输出

**输入样例：**

```
5 //n
1 2 3 2 5
5 //m
ADD 1 4 3
ASK 2 3
ASK 3 5
ADD 2 4 2
ASK 2 5
```

**输出样例：**

```
6
6
8
```

**数据规模**  $m, n \leq 100000$

给出一列数 $\{A_i\}(1 \leq i \leq n)$ ，总共有 $m$ 次操作，操作分如下两种：

- 1.ADD  $x\ y\ z$       将 $x$ 到 $y$ 区间的所有数字加上 $z$
- 2.ASK  $x\ y$         将 $x$ 到 $y$ 区间的最大一个数字输出

关键问题：如何处理1号操作？

每次将区间 $[x,y]$ 包含的所有节点的Max值都加上  $z$   
这样时间复杂度可能会增加到 $O(2n)$

## 数据结构

**struct Node**

{

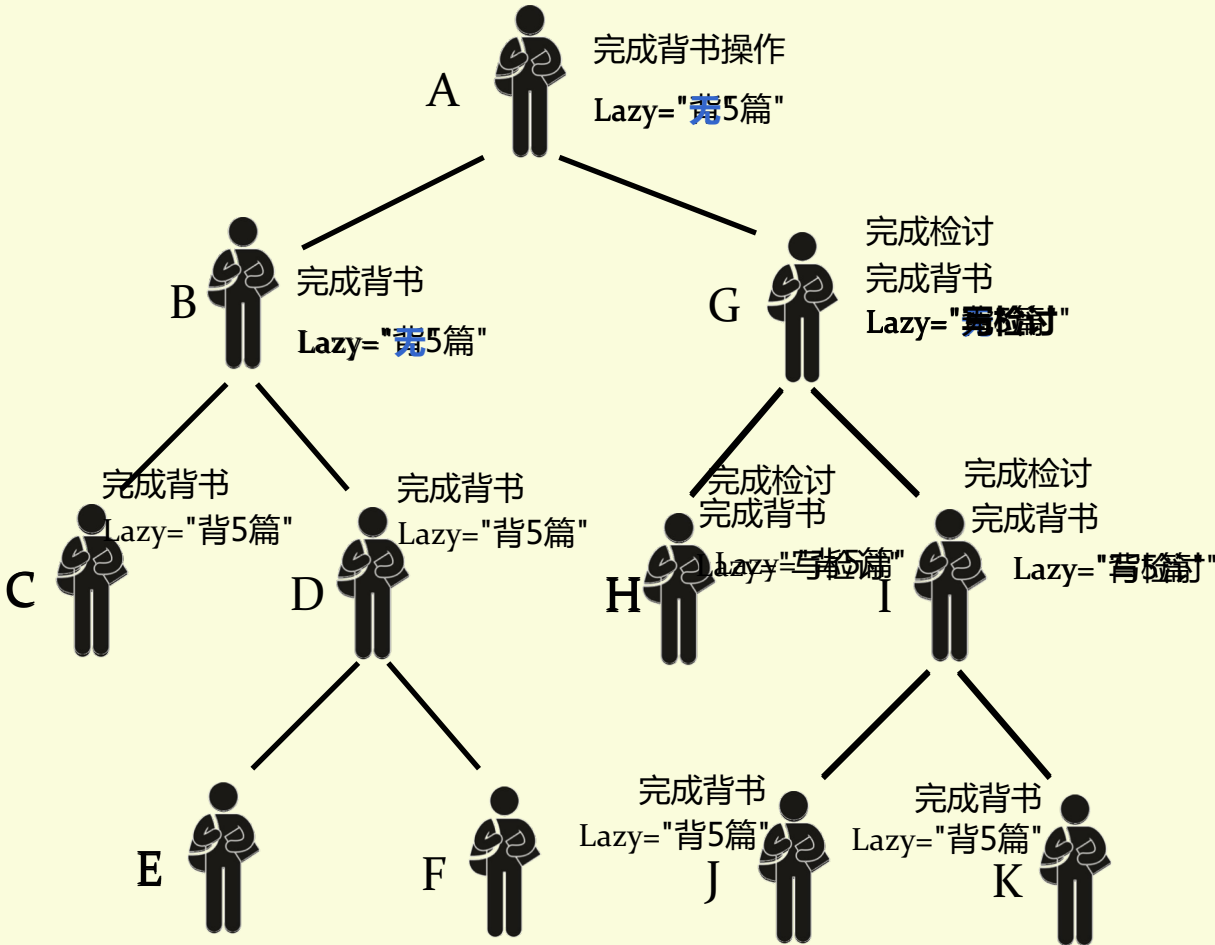
**int a,b,Max ,Lazy ;**        //Max用于记录该节点所表示的区间中的最大值





**}Tree[800001];**

    Lazy标记，是一个延迟标记，记录当前节点接收到的1号操作，但并不真正执行它们，等到需要的时候再说；

# 延迟标记 Lazy

如果需要对一个区间中**每一个叶结点**进行操作,我们不妨先别忙着操作，而是在所有**大区间**上做一个**标记**, 下一次遇到或要用到时，再进行处理(**标记传递**)。



- ①  "A同学，通知所有人，今天背5篇古文"
- ②  "A同学，通知D，我要抽他背古文"
- ③  "A同学，通知G，他所在小组每个同学写一篇1万字的检讨"
- ④  "A同学，通知I，我要检查他的检讨"

# 建树

```
void BuildTree(int p,int x,int y)
{
    Tree[p].a=x;
    Tree[p].b=y;
    Tree[p].Max=Tree[p].Lazy=0;
    if(x==y) {    Tree[p].Max=k[x];    return ;    }
    int Mid=(x+y)>>1;
    BuildTree(p<<1,x,Mid);
    BuildTree(p<<1|1,mid+1,y);
    Tree[p].Max=max(Tree[p<<1].Max,Tree[p<<1|1].Max);
}
```

```
void PutDown(int k)    //下放操作，将累积在点k上的Lazy值下放到它的儿子节点
{
    Tree[ k*2 ].Max  += Tree[k].Lazy;
    Tree[ k*2  ].Lazy += Tree[k].Lazy;
    Tree[ k*2+1 ].Max  += Tree[k].Lazy;
    Tree[ k*2+1 ].Lazy += Tree[k].Lazy;
    Tree[k].Lazy = 0;
}
```

## 修改操作

```
void Change(int x,int y,int k,int z) //将区间x到y的所有数字增加z
{
    if( Tree[k].Lazy != 0 ) PutDown(k); //遇到延迟标记，先进行处理
    if( x<=Tree[k].a && Tree[k].b<=y)
    {
        Tree[k].Lazy +=z;    //标记节点要增加的值，也意味着它的子孙节点都要增加相同的值
        Tree[k].Max  +=z;
        return ;
    }
    int Mid=(Tree[k].a+Tree[k].b)>>1;
    if( x<=Mid && y>=Tree[k*2].a ) Change(x,y,k*2,z);
    if( y > mid && x<=Tree[k*2+1].b) Change(x,y,k*2+1,z);
    Tree[k].Max = max(Tree[k*2].Max,Tree[k*2+1].Max);
}
```



## 询问操作

```
int Ask(int x,int y,int k)    //询问区间[x,y]的最大值，当前讨论到k号点了。
{
    if( Tree[k].Lazy != 0) PutDown(k);    //只要k点累积有Lazy值，就下放

    if(x<=Tree[k].a && Tree[k].b<=y)    return Tree[k].Max;

    int  Lmax=0 , Rmax=0;
    int  Mid=(Tree[k].a+Tree[k].b)>>1;
    if(x<=Mid && y>=Tree[k*2].a) Lmax=Ask(x,y,k*2);
    if( y > mid && x<=Tree[k*2+1].b)    Rmax=Ask(x,y,k*2+1);
    return max(Lmax, Rmax);
}
```

**Lazy思想**：对整个结点进行的操作，先在结点上做标记，而并非真正执行，直到根据查询操作的需要分到下层。

## 例2 涂色 NKOJ2295

**题目：**在数轴上进行一系列操作。每次操作有两种类型，一种是在线段 $[a,b]$ 上涂上颜色，另一种将 $[a,b]$ 上的颜色擦去。问经过一系列的操作后，有多少条单位线段 $[k,k+1]$ 被涂上了颜色。

**输入：**

第一行，一个整数 $n$  ( $n \leq 1000$ )。

接下来 $n$ 行，每行三个整数

第一个数为1表示涂色，0表示擦去。

第二、三个数表示线段 $[a,b]$  ( $1 \leq a < b \leq 2000$ )。

**输出：**有多少条单位线段 $[k,k+1]$ 被涂上了颜色。

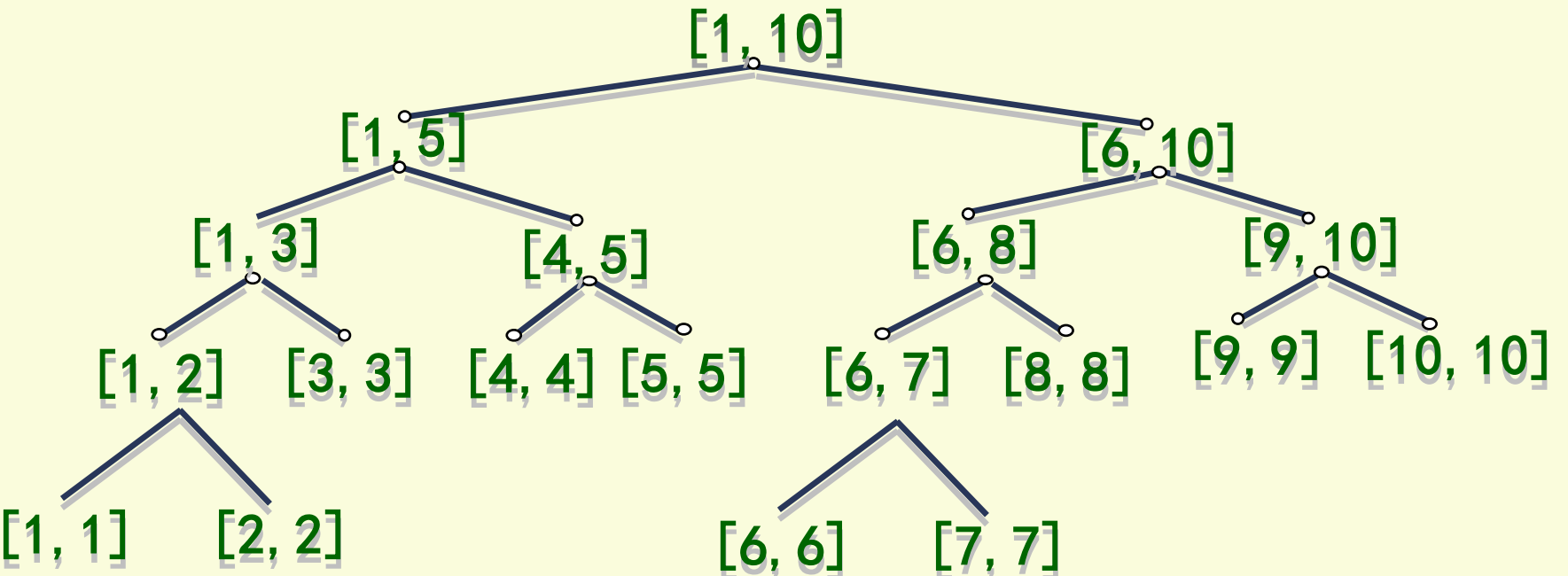
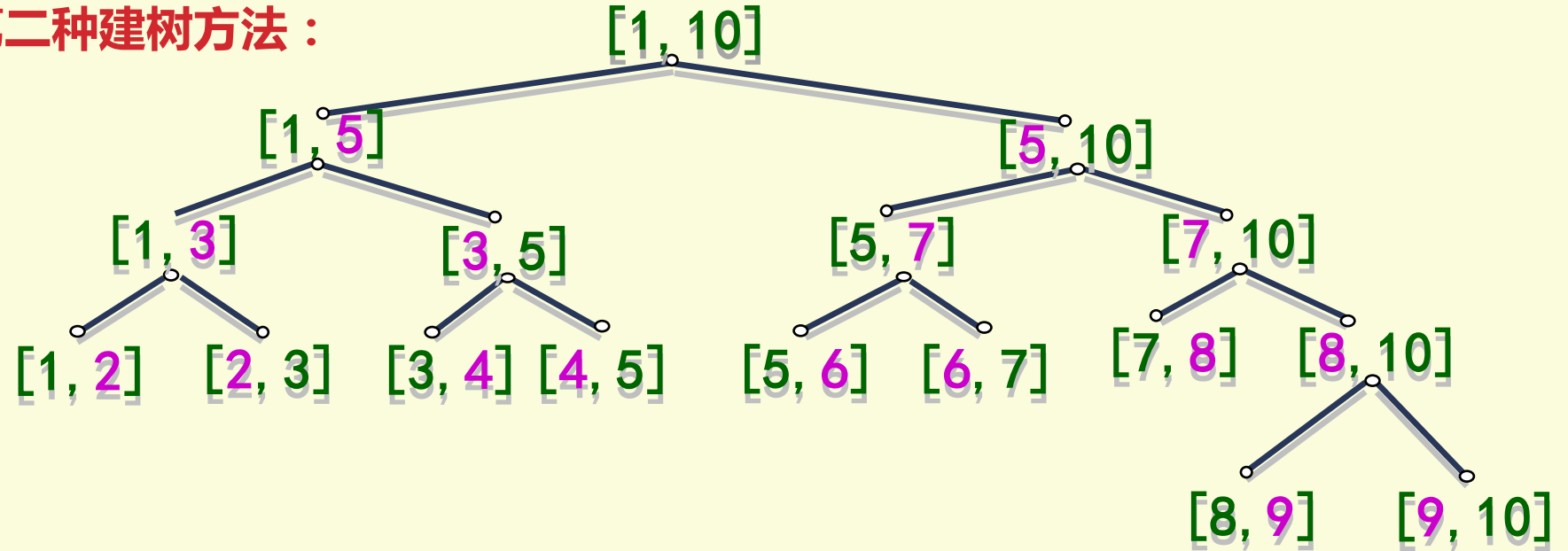
**样例输入：**

```
5
1 1 15
0 4 9
1 7 18
1 7 9
0 1 3
```

**样例输出：**

```
11
```

第二种建树方法：



# 建树

```
struct node{
    int a,b,Lazy;    //Lazy=1表示擦除, Lazy=2表示涂色
}Tree[maxn];
int n,p=0;

void BuildTree(int p,int x,int y)
{
    Tree[p].a=x;    Tree[p].b=y;
    Tree[p].Lazy=0;
    if (x+1<y)
    {
        int Mid=(x+y)>>1;
        BuildTree(p<<1,x,Mid);
        BuildTree((p<<1)+1,Mid,y);
    }
}
```

**Lazy思想**：对整个结点进行的操作，先在结点上做标记，而并非真正执行。

```
void PutDown_Clean(int k)    //Tree[k].Lazy=1,下放擦除操作
{
    Tree[k].Lazy=0;
    Tree[k*2].Lazy=1;
    Tree[k*2+1].Lazy=1;
}
```

```
void PutDown_Paint(int k)    //Tree[k].Lazy=2,下放涂色操作
{
    Tree[k].Lazy=0;
    Tree[k*2].Lazy=2;
    Tree[k*2+1].Lazy=2;
}
```

# 删除操作

```
void Delete(int k,int x,int y) //擦除区间[x,y]的颜色，当前讨论k号点
{
    if(Tree[k].Lazy==1) return; //若k号点已被擦除，结束操作
    if( (x<=Tree[k].a) && (y>=Tree[k].b) ) //若k号点被完全覆盖
    {
        Tree[k].Lazy=1; //将k号点标记为已被擦除
        return;
    }
```

//讨论区间[x,y]与k号点部分相交的情况

```
if(Tree[k].a+1<Tree[k].b) //若k号点存在左右儿子，则讨论之
{
    if(Tree[k].Lazy==2) PutDown_Paint(k);
    int Mid=(Tree[k].a+Tree[k].b)>>1;
    if(x<=Mid && y>Tree[k*2].a) Delete(k*2,x,y);
    if(y>=Mid && x<Tree[k*2+1].b) Delete(k*2+1,x,y);
}
}
```

若k号点没有左右儿子，也就是它本身表示一条单位线段，  
那么与[x,y]相交的情况只有包含和不相交两种，不会出现部分相交。

## 插入操作与删除操作非常相似

```
void Insert(int k,int x,int y)    //给区间[x,y]涂色，当前讨论k号点
{
    if(Tree[k].Lazy==2) return;    //若k号点已被涂色，结束操作
    if ((x<=Tree[k].a)&&(y>=Tree[k].b))    //若k号点被完全覆盖
    {
        Tree[k].Lazy=2;    //将k号点标记为已涂色
        return;
    }
    //讨论区间[x,y]与k号点部分相交的情况
    if(Tree[k].a+1<Tree[k].b)    //若k号点存在左右儿子，则讨论之
    {
        if(Tree[k].Lazy==1) PutDown_Clean(k);
        int Mid=(Tree[k].a+Tree[k].b)>>1;
        if(x<=Mid && y>Tree[k*2].a) Insert(k*2,x,y);
        if(y>=Mid && x<Tree[k*2+1].b) Insert(k*2+1,x,y);
    }
}
```



# 获取答案

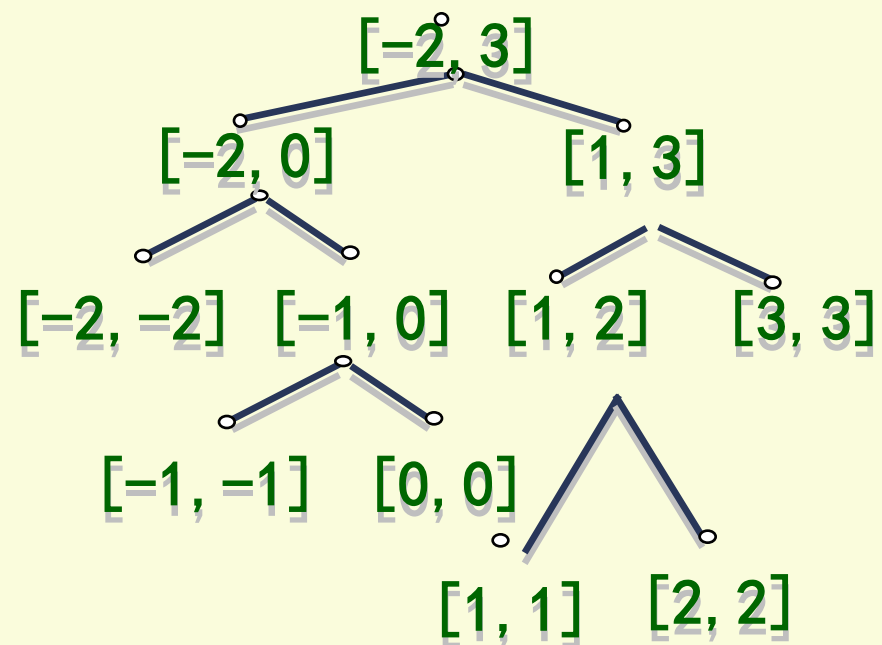
```
int GetAns(int p) //从根节点开始讨论，当前到了p号点
{
    if(Tree[p].Lazy==1) return 0; //该点被标记为了已擦除，结束
    if(Tree[p].Lazy==2) return (Tree[p].b-Tree[p].a); //被标记为已涂色
    if(Tree[p].a+1<Tree[p].b) //讨论左右儿子的情况，存在左右儿子
    {
        int t1=GetAns(p*2);
        int t2=GetAns(p*2+1);
        return (t1+t2);
    }
}
```

## 小结

线段树可高速进行区间的插、删、改、询。  
延迟标记lazy，是线段树的精髓，目的是为了减少操作次数，提高线段树的效率。

Lazy标记真是巧；  
事情来了它阻挠；  
要到用时才去做；  
效率提高真不少。

## 特殊的区间



对于区间  $[a, b]$  若  $a+b \geq 0$  那么

左儿子表示的区间为  $[a, (a+b)/2]$

右儿子表示的区间为  $[(a+b)/2+1, b]$

对于区间  $[a, b]$  若  $a+b < 0$  那么

左儿子表示的区间为  $[a, (a+b)/2-1]$

右儿子表示的区间为  $[(a+b)/2, b]$

作业：2297,2295,1887,1906