线段树、 树状数组

邓丝雨

引入1：

- 有n（n<=50000）个数，m（m<=50000）次询问，
- 每次询问区间Li到Ri的数的和
- 要求输出每一次询问的结果……

引入2： RMQ (Range Minimum/Maximum Query)问题

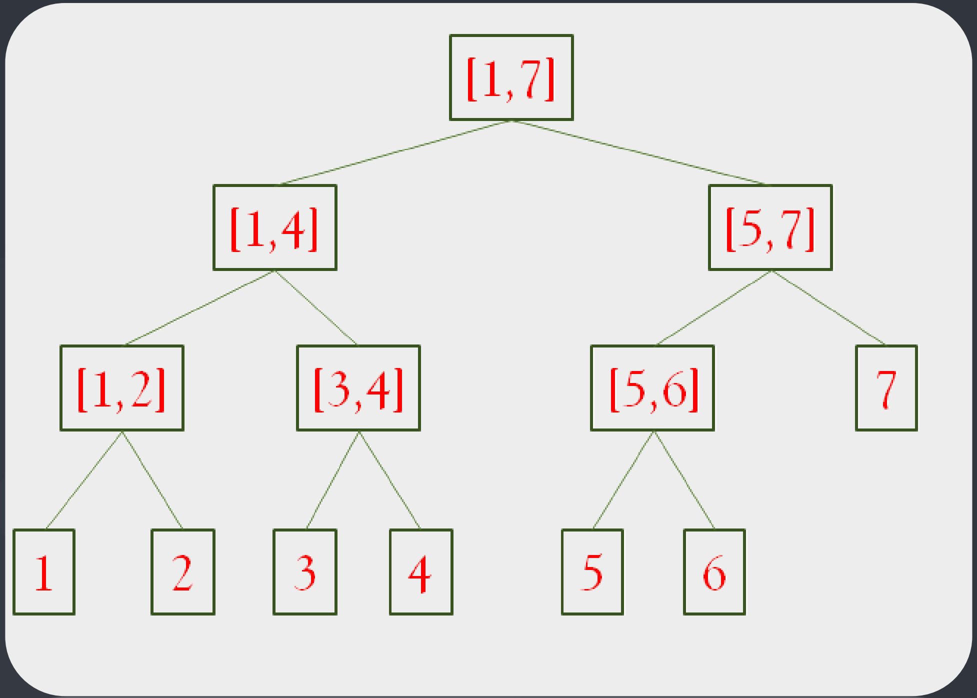
- 有n（n<=50000）个数，m（m<=50000）次询问，
- 每次询问区间L到Ri的数的最大值

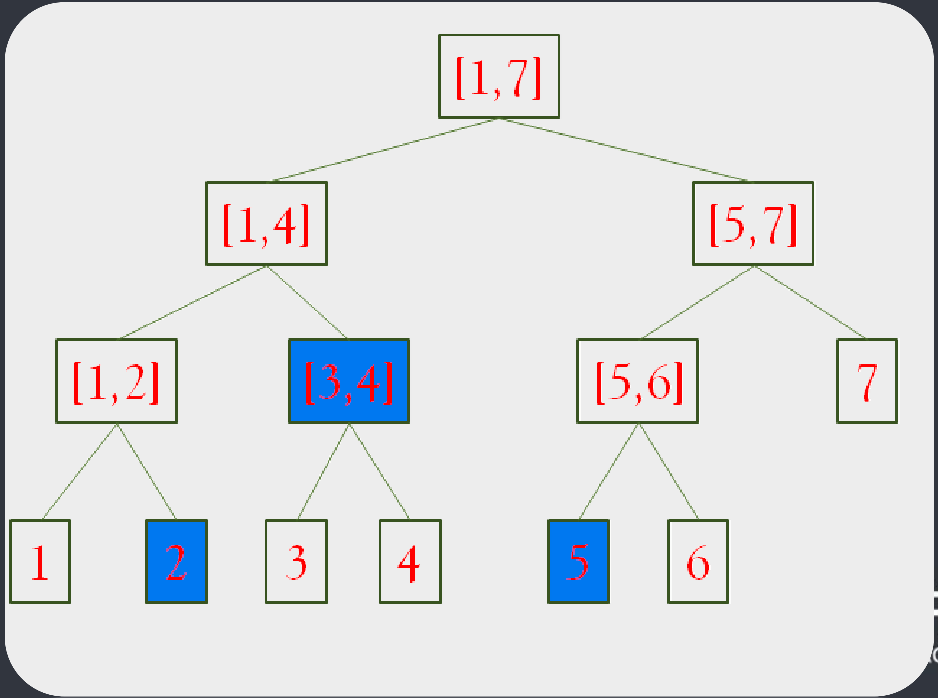
线段树

- 线段树是用一种树状结构来存储一个连续区间的信息的数据结构。
- 它主要用于处理一段连续区间的插入,查找,统计,查询等操作。
- 复杂度： 设区间长度是n,所有操作的复杂度是logn级别。

线段树

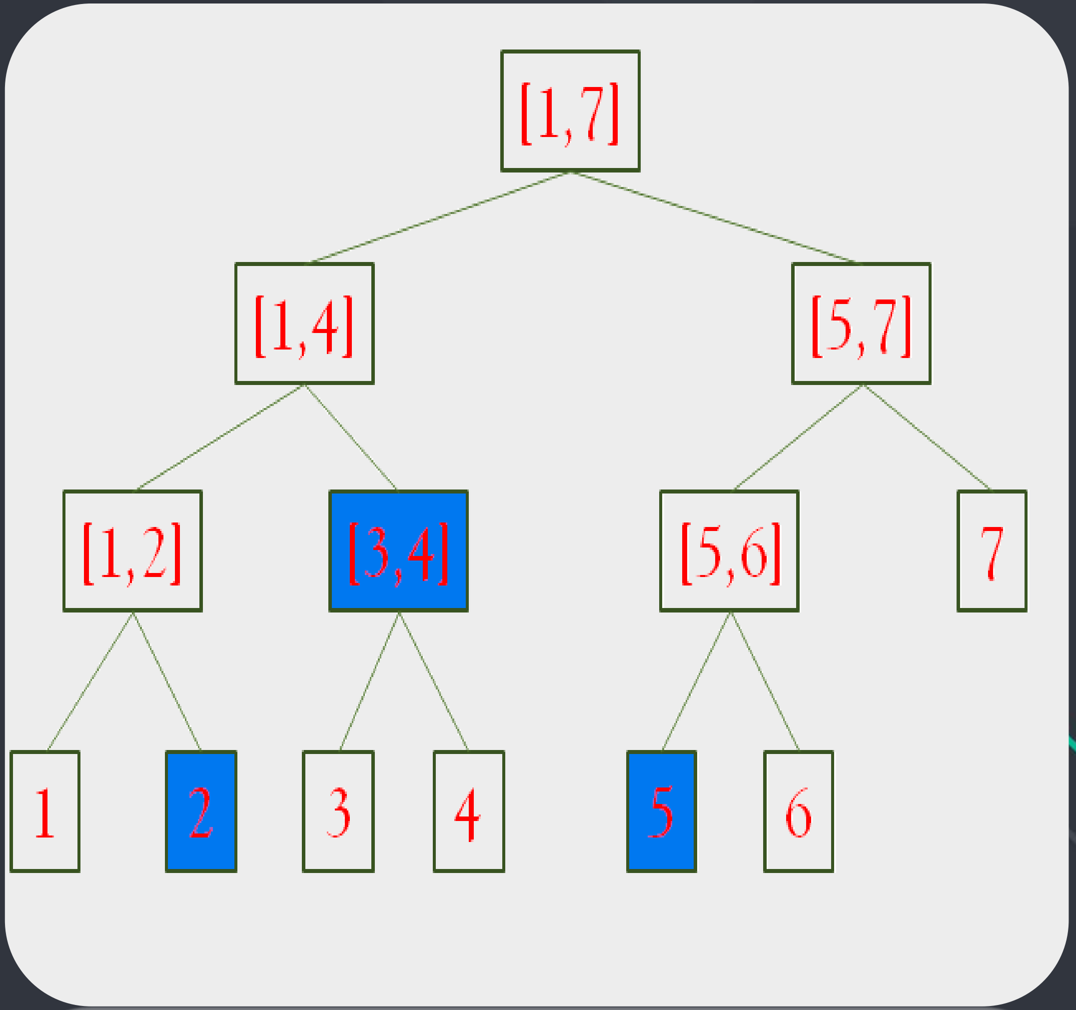
- 线段[1, 7]的线段树 和[2, 5]的分解





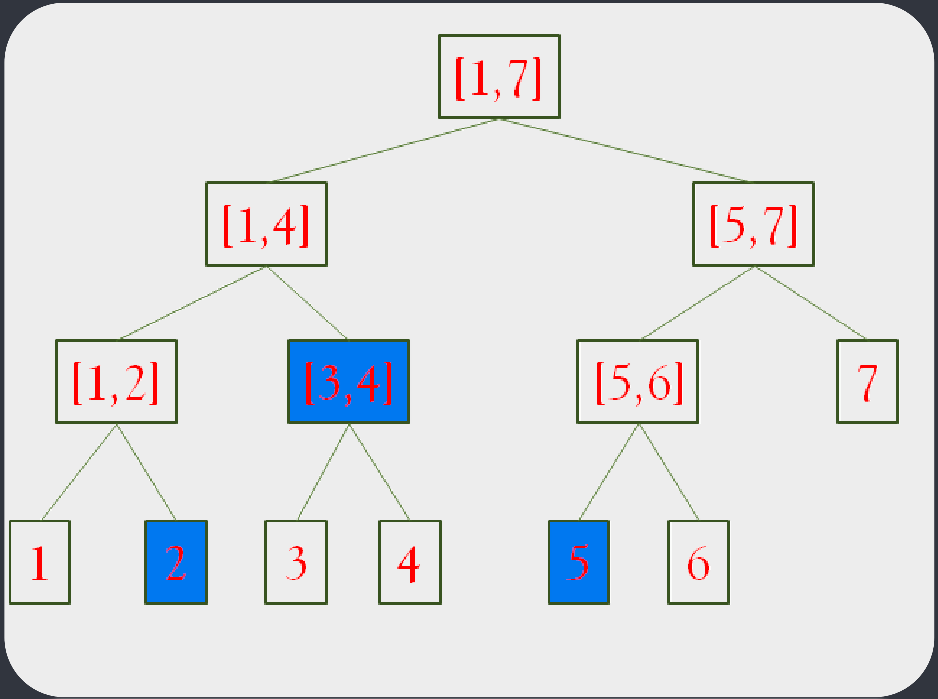
线段树的几点性质

- ◈ 线段树是平衡的2叉树,最大深度logn(n为线段树所表示区间的长度)
- ◈ 树中的每一个节点代表对应一个区间（叶子节点是一个点……)
- ◈ 每个节点（所代表的区间）完全包含它的所有子孙节点
- ◈ 对于任意两个节点（所代表的区间）：要么完全包含，要么互不相交



线段树的几点性质

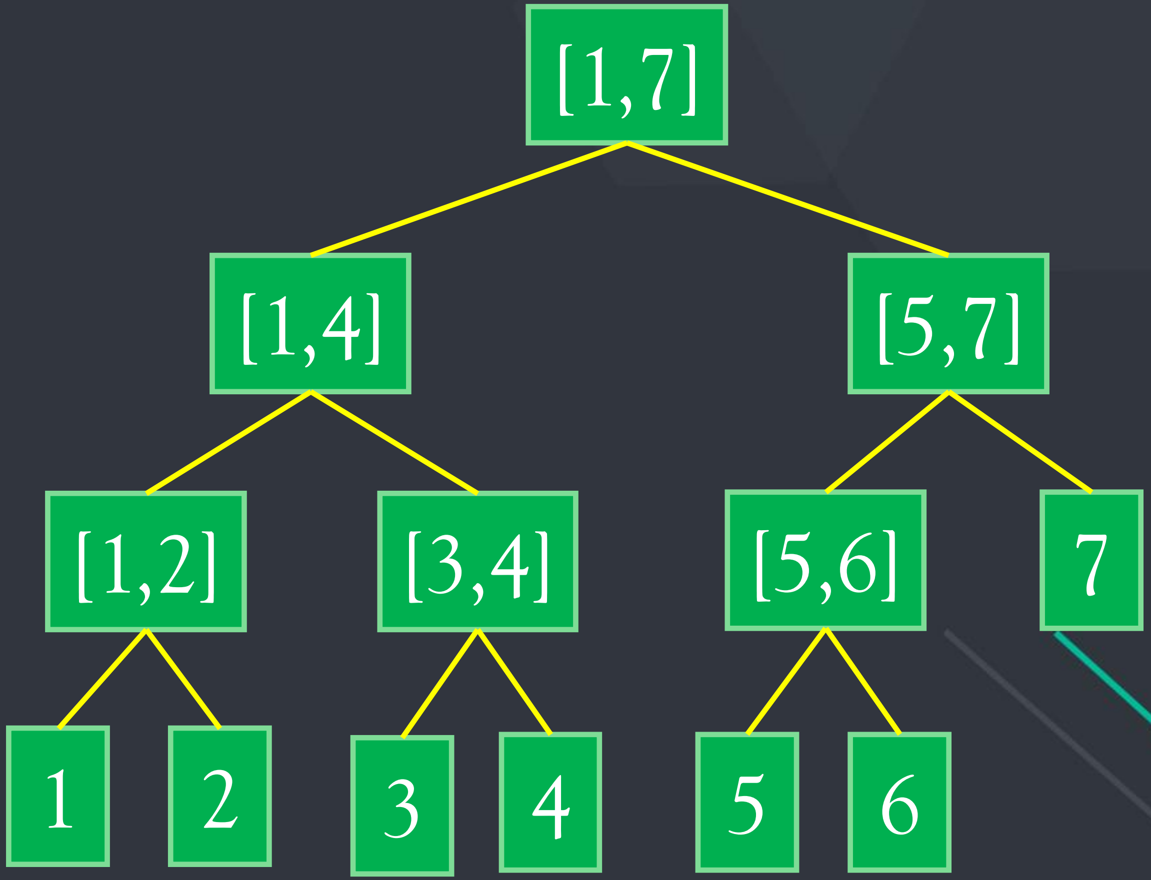
- ◇ 在进行区间操作和统计时把区间等价转换成若干个子区间(logn个)的相同操作。
- ◇ 任意的线段[a,b]在线段树的查询或查找过程中把这个线段最多分成log(b-a)份（显然每一层最多2个区间）
- ◇ So，线段树除建树外的操作都是log(n)级别的复杂度。



线段树的运用

- 线段树的每个节点上往往都增加了一些其他的域。在这些域中保存了某种动态维护的信息，视不同情况而定。这些域使得线段树具有极大的灵活性，可以适应不同的需求。

- 用线段树来维护每个区间的和
- For example:
- 7个数分别为
- 1 4 5 6 3 2 7



例1：

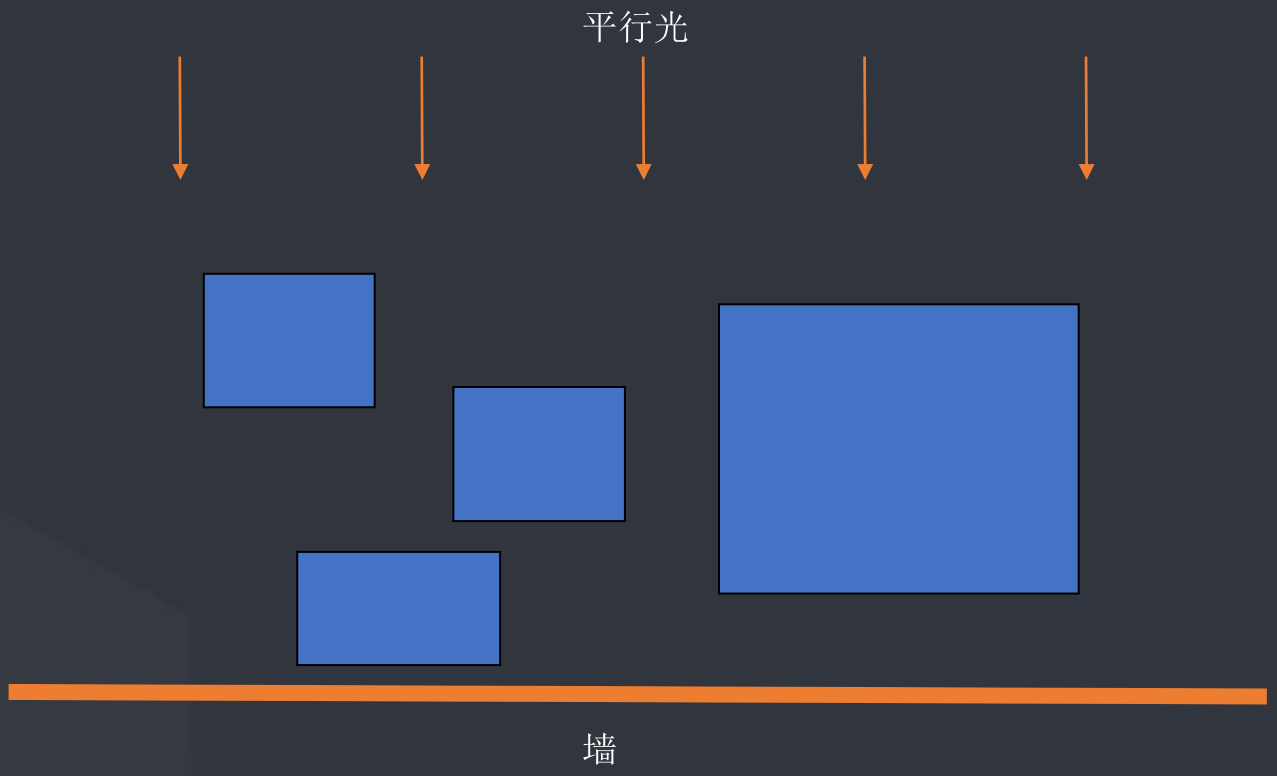
- 给你一个长度为n的序列
- 有如下操作
- 将第i个数加或减x
- 求区间Li到Ri的和

```
void build(int p, int l, int r)
{
        if (l == r) {tree[p] = a[l]; return;}
        int mid = (l + r) / 2;
        build(p * 2, l, mid);
        build(p * 2 + 1, mid + 1, r);
        tree[p] = tree[p * 2] + tree[p * 2 + 1];
}
void change(int p, int l, int r, int x, int num)
{
        if (l == r) {tree[p] += num; return;}
        int mid = (l + r) / 2;
        if (x <= mid) change(p * 2, l, mid, x, num);
        else change(p * 2 + 1, mid + 1, r, x, num);
        tree[p] = tree[p * 2] + tree[p * 2 + 1];
}
int find(int p, int l, int r, int x, int y)
{
        if (x <= l && r <= y) return tree[p];
        int mid = (l + r) / 2;
        if (y <= mid) return find(p * 2, l, mid, x, y);
        if (x > mid) return find(p * 2 + 1, mid + 1, r, x, y);
        return (find(p * 2, l, mid, x, mid) + find(p * 2 + 1, mid + 1, r, mid + 1, y));
}
```

- 线段树很关键的一点：需要维护哪些区间的附加信息，怎样维护这个信息！
- 比如维护区间和，区间最大（小）值，等等。
- 关键是这个“等等”究竟是什么？

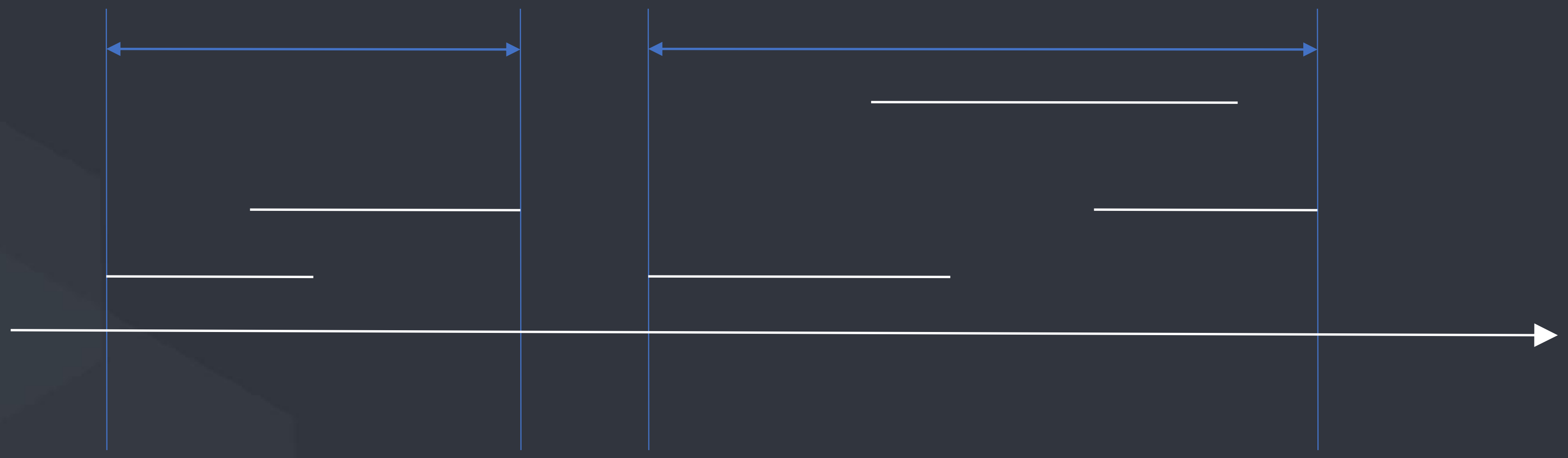
例2：

- 桌子上零散地放着若干个盒子，桌子的后方是一堵墙。如右图所示。现在从桌子的前方射来一束平行光， 把盒子的影子投射到了墙上。问影子的总宽度是多少？



例2:

- 这道题目是一个经典的模型。在这里，我们略去某些处理的步骤，直接分析重点问题，可以把题目抽象地描述如下：x轴上有若干条线段，求线段覆盖的总长度。

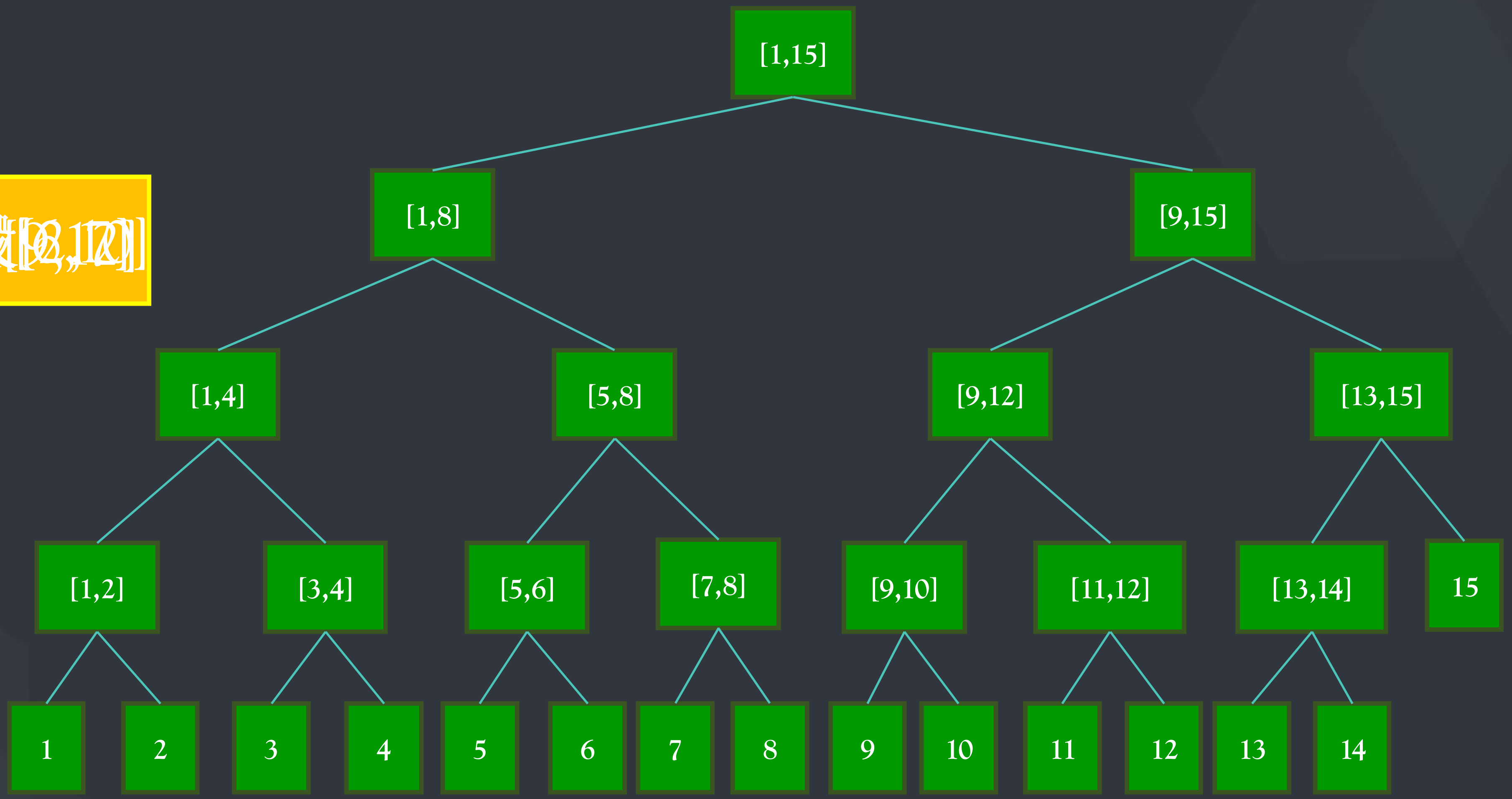


- ◈ 维护什么信息呢?
- ◈ 当前区间内被覆盖的长度是必然要维护的
- ◈ 但是如果只维护区间内被覆盖的长度，我们添加线段的时候就需要访问每一个叶子节点。。。于是复杂度是O（L*logL）
- ◈ 添加线段比朴素的时候还慢啊有木有……
- ◈ 这样的算法不被卡什么算法被卡?

- 给线段树每个节点增加一个附加信息cover。cover=1表示该结点所对应的区间被完全覆盖，cover=0表示该结点所对应的区间未被完全覆盖。
- 一旦当前区间已经被完全覆盖了，就不用更改他的儿子们的标记了……

例2：

插入线段[2,12]

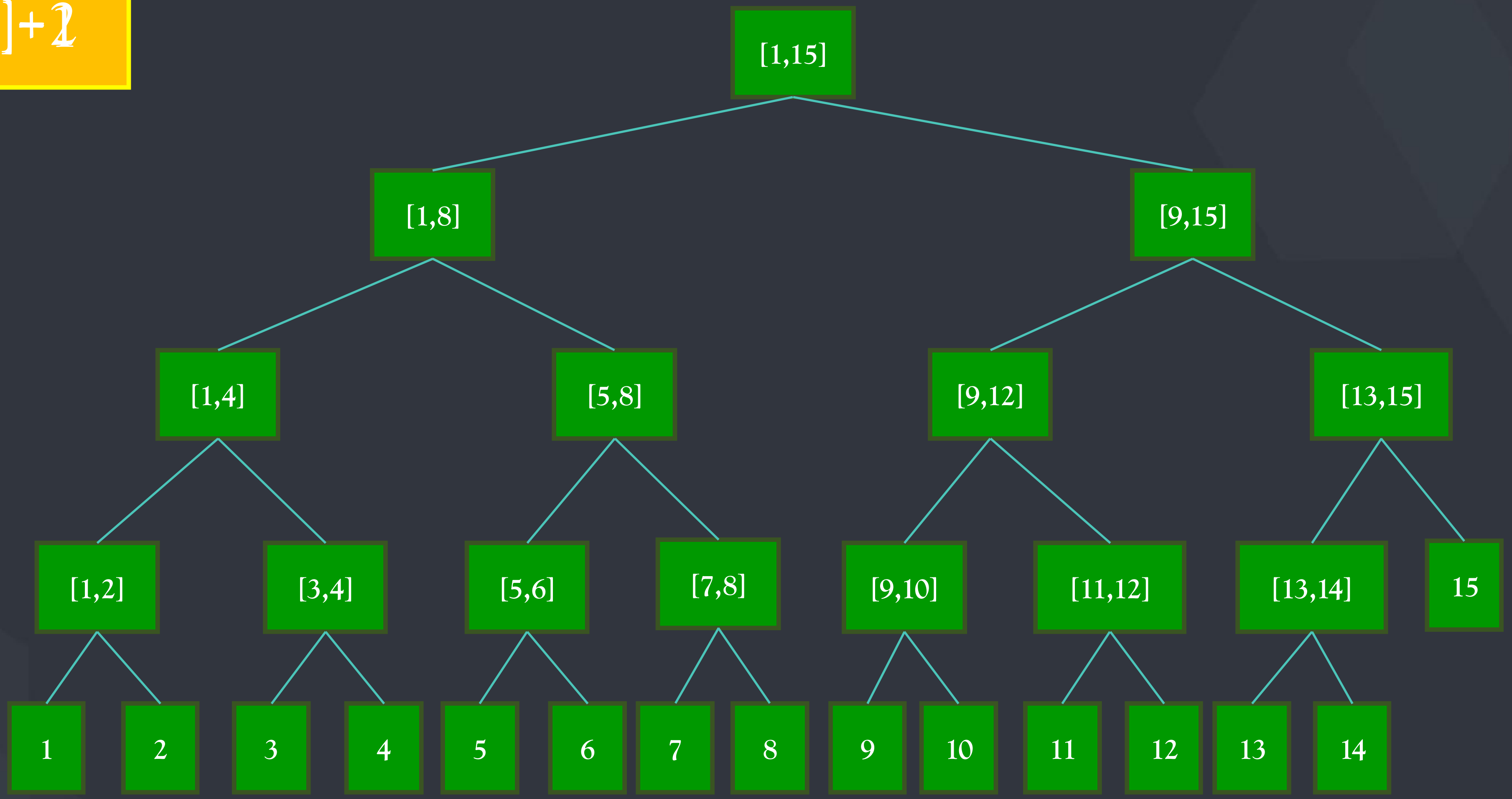


[1,15]
[1,8]
[9,15]
[1,4]
[5,8]
[9,12]
[13,15]
[1,2]
[3,4]
[5,6]
[7,8]
[9,10]
[11,12]
[13,14]
15
1
2
3
4
5
6
7
8
9
10
11
12
13
14

例3:

- 给你N个数，Q个操作，操作有两种，
- ‘Q a b ’ 是询问a~b这段数的和，
- ‘C a b c’ 是把a~b这段数都加上c。

区间[6,7]+2


[1,15]
[1,8]
[9,15]
[1,4]
[5,8]
[9,12]
[13,15]
[1,2]
[3,4]
[5,6]
[7,8]
[9,10]
[11,12]
[13,14]
15
1
2
3
4
5
6
7
8
9
10
11
12
13
14

例3：

- 什么时候更新子区间？？
- 当已经标记过的区间有部分被更改——那么这个区间的整体更改就不是原来那么多了，标记就向下传一层
- 注意：只传一层，如果这一层的区间还有更改的话再往下传

```cpp
void build(int p, int l, int r)
{
    lazy[p] = 0;
    if (l == r) {tree[p] = a[l]; return;}
    int mid = (l + r) >> 1;
    build(p * 2, l , mid);
    build(p * 2 + 1, mid + 1, r);
    tree[p] = tree[p * 2] + tree[p * 2 + 1];
}
void pushdown(int p,int tot)
{
    lazy[p * 2 + 1] += lazy[p];
    lazy[p * 2] += lazy[p];
    tree[p * 2] += (tot - tot / 2) * lazy[p];
    tree[p * 2 + 1] += (tot / 2) * lazy[p];
    lazy[p] = 0;
}
void add(int p, int l, int r, int x, int y, int z)
{
    if (x <= l && r <= y)
    {
        lazy[p] += z;
        tree[p] += ((long long)(r - l + 1)) *z;
        return;
    }
    pushdown(p, r - l + 1);
    int mid = (l + r) >> 1;
    if (x <= mid) add(p * 2, l ,mid, x, y, z);
    if (y > mid) add(p * 2 + 1, mid + 1, r, x, y, z);
    tree[p] = tree[p * 2] + tree[p * 2 + 1];
}
```

```cpp
long long  find(int p, int l, int r, int x, int y)
{
    if (x <= l && r <= y) return tree[p] ;
    int mid = (l + r) >> 1;
    long long re = 0;
    if (lazy[p] != 0) pushdown(p, r - l + 1);
    if (x <= mid) re += find(p * 2, l ,mid, x, y);
    if (y > mid) re += find(p * 2 + 1, mid + 1, r, x, y);
    return re;

}
```

例4：

- 区间平方和问题：给你一个长度为N的序列，之后有M次操作，操作包括两类：
- A L R 为给区间[L,R]的所有数都加上一个A的值
- S L R 为求区间[L,R]的所有数的平方的和

例5:

- 如题，已知一个数列，你需要进行下面三种操作：
- 1.将某区间每一个数乘上x
- 2.将某区间每一个数加上x
- 3.求出某区间每一个数的和

例6:

- 给你n个数，然后q次询问，t=0的时候把区间内的值都开根取整，t=1的时候就输出区间的和

例7：

- 给出n个点的平面二维坐标，对于每个坐标，如果这个坐标跟(0,0)形成的矩形内包含的点数为 k （包含边界，但不包含坐标本身），那么这个坐标就是 level k。输出level 0 – leveln-1的点数分别是多少。
- n个点按照纵坐标y升序给出

例7：

- 由于题目预先给出的坐标已经按照纵坐标y排序，所以线段树就建立在x上。然后按照给出点的顺序，我们先询问当前(0,x)区间有多少个点，再把这个点放到线段树里面（因为统计点数的时候不包含本身）。
- 这里很巧妙的是，因为放的点按照y排序，所以先放进线段树的点的纵坐标肯定小于当前询问的点，而且询问的区间是(0,x)的话，那么得到的点数就是就是对应矩形区域内的点数。

```
void build(int p, int l, int r, int x)
{
    if (l == r){tree[p]++; return;}
    int mid = (l + r) / 2;
    if (mid >= x) build(p * 2, l, mid, x);
    if (mid < x) build(p * 2 + 1, mid + 1, r, x);
    tree[p] = tree[p * 2] + tree[p * 2 + 1];
}
int find(int p, int l, int r, int x, int y)
{
    if ((x <= l) && (r <= y)) return tree[p];
    int mid = (l + r) / 2;
    if (mid >= y) return find(p * 2, l, mid, x, y);
    if (mid < x) return find(p * 2 + 1, mid + 1, r, x, y);
    int t = find(p * 2, l, mid, x, y)
            +find(p * 2 + 1, mid + 1, r, x, y);
    return t;
}
```

```
int main()
{
    //freopen("poj2352.in","r",stdin);
    //freopen("poj2352.out","w",stdout);
    scanf("%d", &n);
    for (int i = 1; i <= n; i++)
    {
        int y;
        scanf("%d%d", &b[i], &y);
        if (b[i] > maxn) maxn = b[i];
    }
    for (int i=1;i<=n;i++)
    {
        build(1, 0, maxn, b[i]);
        a[find(1, 0, maxn, 0, b[i]) - 1]++;
    }
    for (int i = 0; i != n; ++i)
        printf("%d\n", a[i]);

    return 0;
}
```

例8:

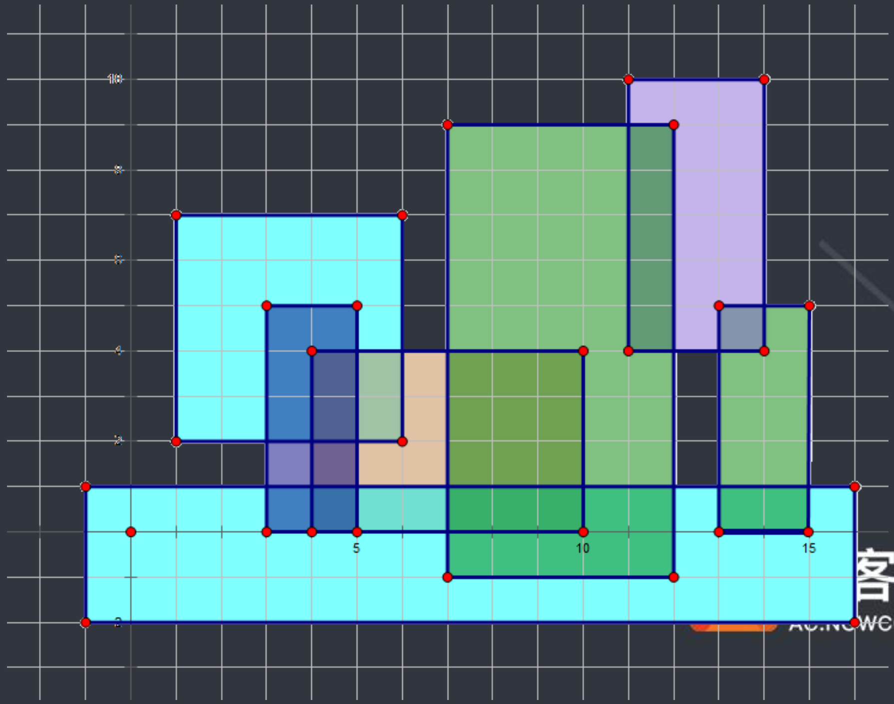
- 用线段树求逆序对数

例9:

- 有一个包含n个元素的数组，要求实现以下操作：
  DELETE k：删除位置k上的数。右边的数往左移一个位置。
  QUERY i j：查询位置i~j上所有数的最小值和最大值。

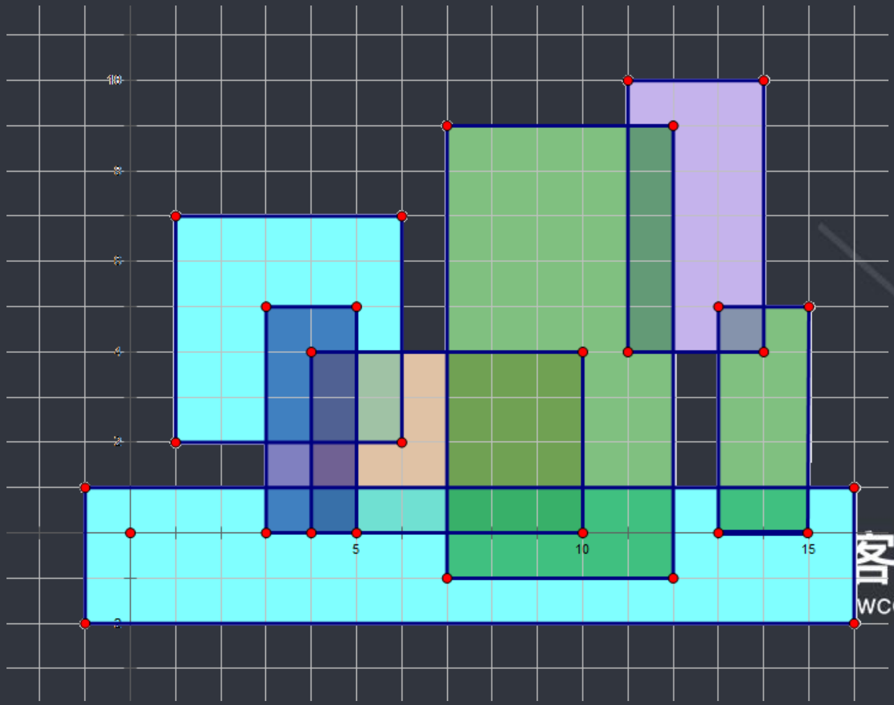
例10：

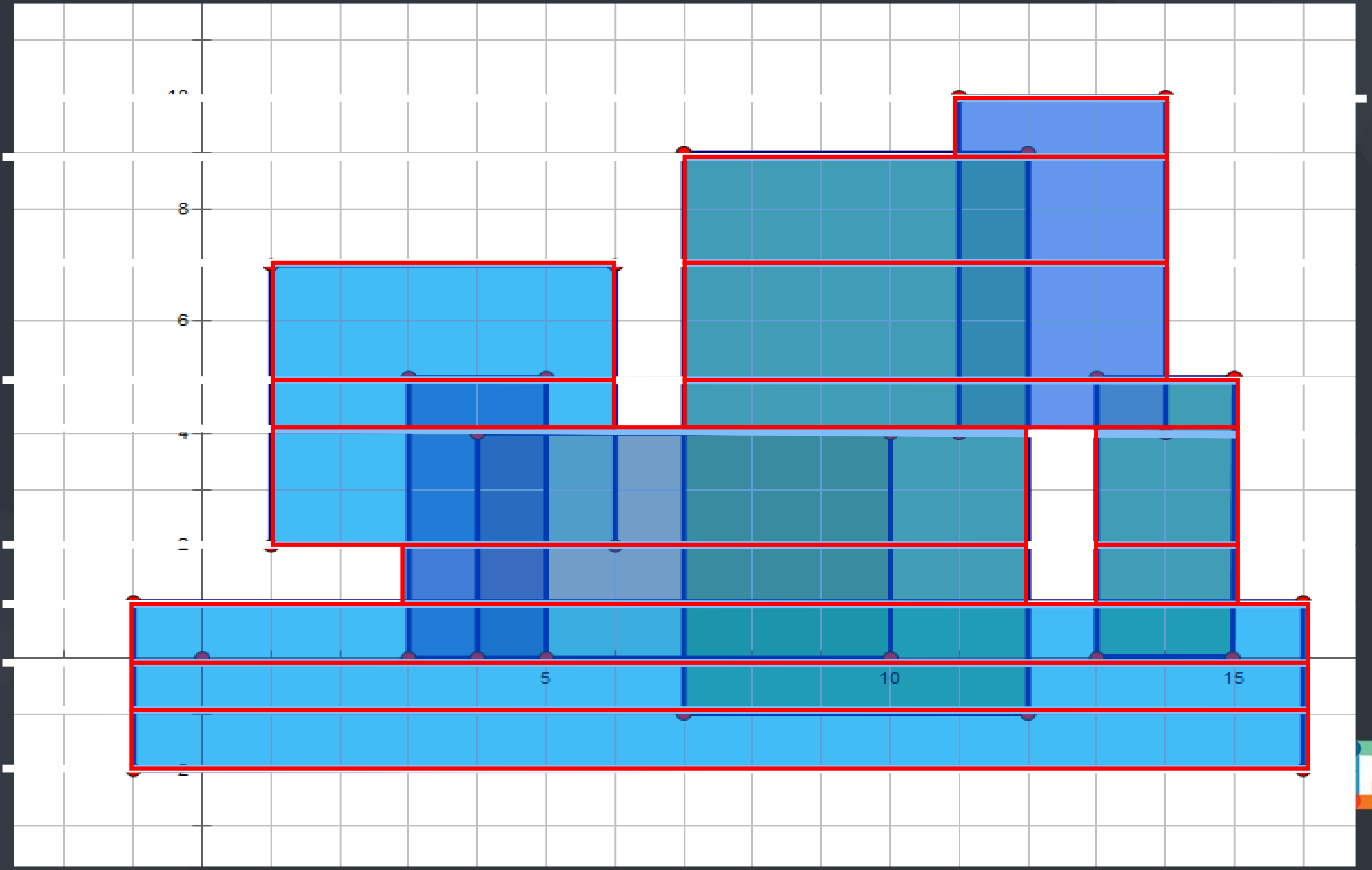
- 给出平面坐标系上若干个长方体的左上角和右下角的坐标，求这些矩形的互相覆盖后的面积……



例10：

- 给出平面坐标系上若干个长方体的左上角和右下角的坐标，求这些矩形的互相覆盖后的面积……





8
6
5
10
15

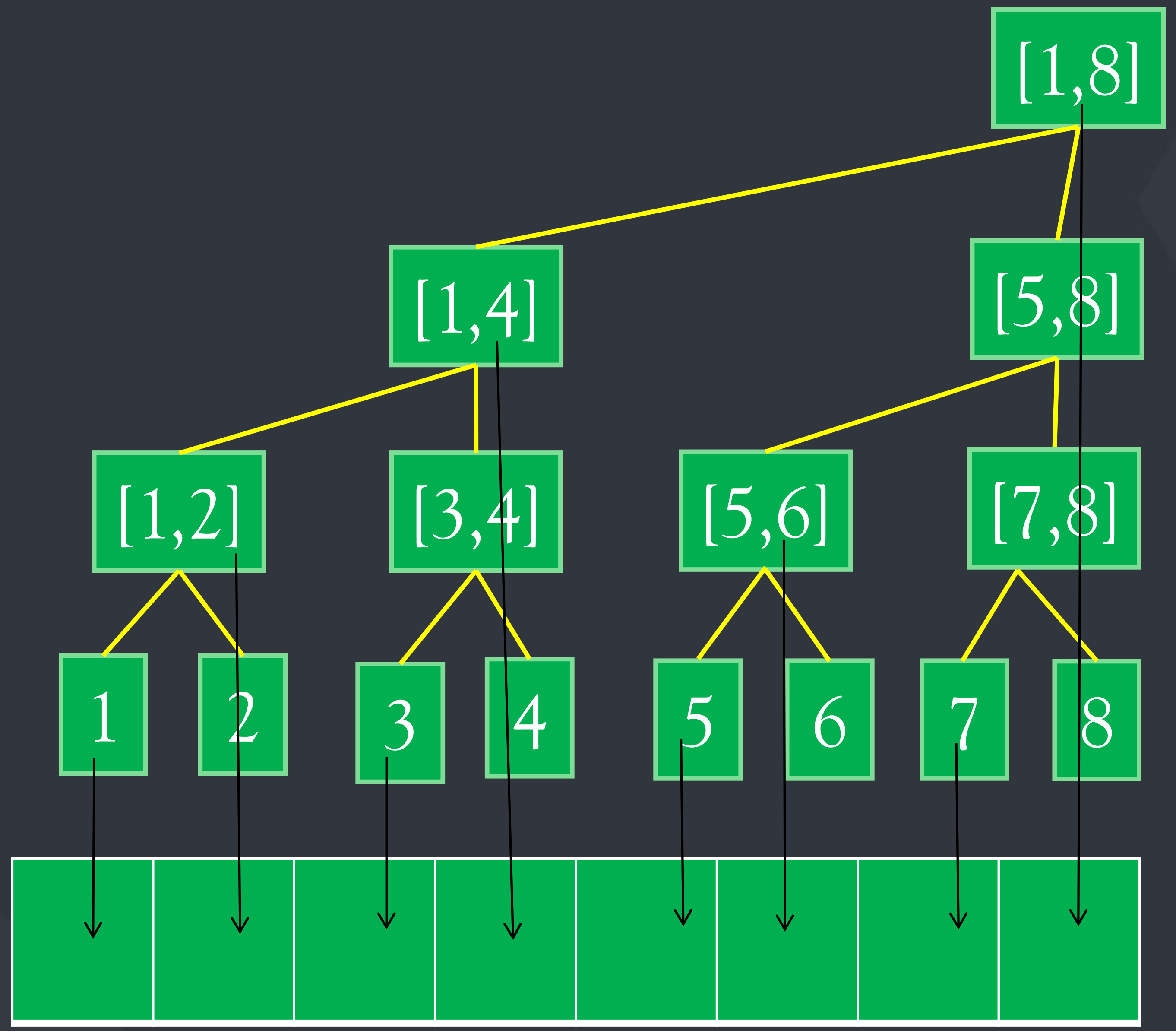
```
    while(t--)
    {
        scanf("%d", &n);
        m = 1;
        for(int i = 1; i <= n; i++)
        {
            scanf("%lf%lf%lf%lf", &x1, &y1, &x2, &y2);
            line[m].x = x1;
            line[m].y1 = y1;
            line[m].y2 = y2;
            line[m].f = 1;
            y[m] = y1;
            m++;
            line[m].x = x2;
            line[m].y1 = y1;
            line[m].y2 = y2;
            line[m].f = -1;
            y[m] = y2;
            m++;
        }
                sort(line + 1 , line + m, cmp);
        sort(y + 1,y + m);
        build(1, 1, m - 1);

        change(1, 1, m - 1, line[1]);
        double sum = 0;
        for(int i = 2; i < m; i++)
        {
                //printf("%d %.2lf\n",i, tree[1].cnt2);
            sum += tree[1].cnt2 * (line[i].x - line[i - 1].x);

            change(1, 1, m - 1, line[i]);
        }

        printf("%.2f\n",sum);
    }
```

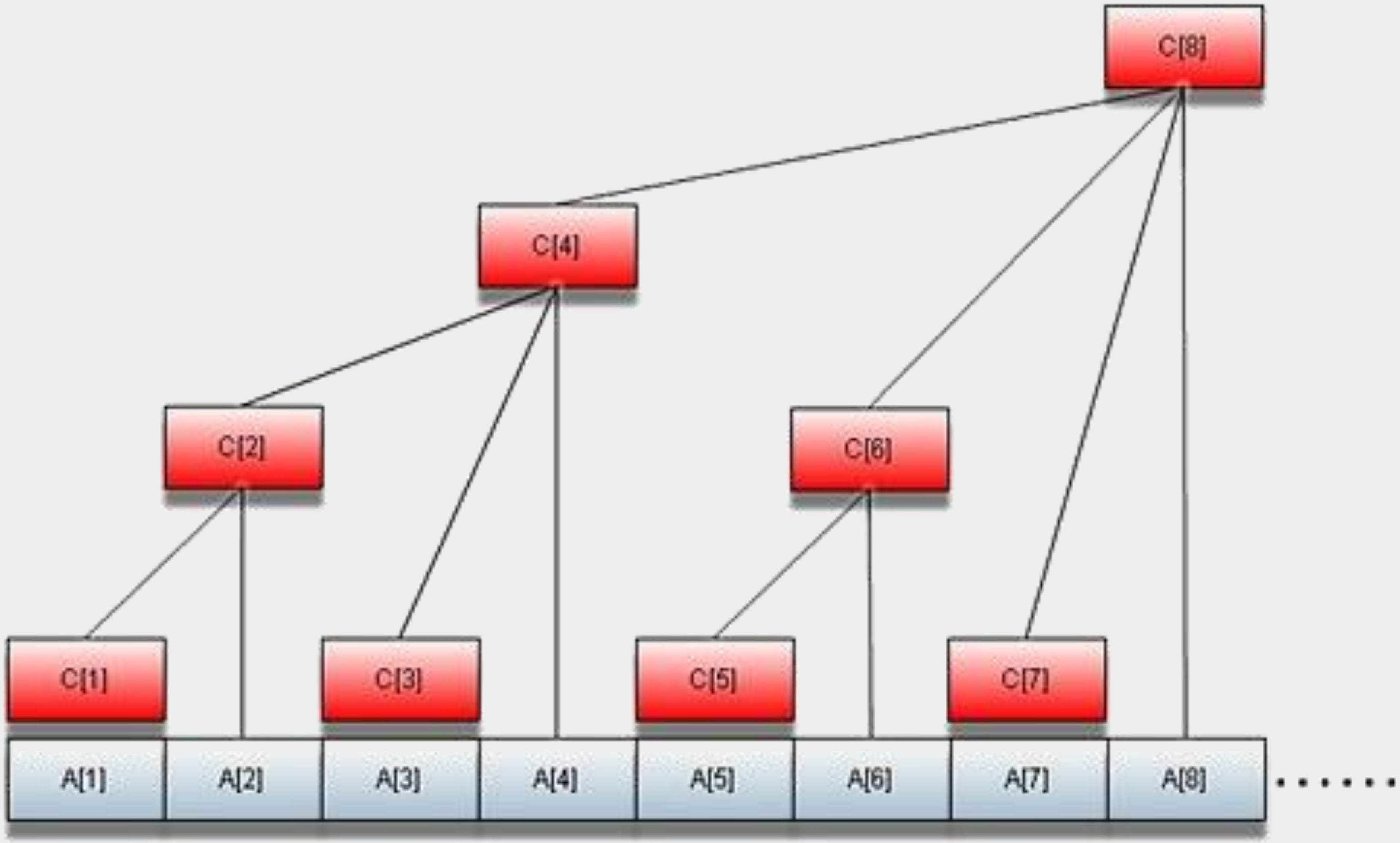
```
void build(int p,int l,int r)
{
    tree[p].cnt= tree[p].c = 0;
    tree[p].lf = y[l];
    tree[p].rf = y[r];
    if(l + 1 == r)  return;
    int mid = (l + r) >> 1;
    build(p * 2, l, mid);
    build(p * 2 + 1 ,mid, r);
}
void change(int p, int l, int r, ty line)
{
        if(line.y1 == tree[p].lf && line.y2 == tree[p].rf)
    {
        tree[p].c += line.f;
        calc(p, l, r);
        return;
    }
    int mid =(l + r) / 2;
    if(line.y2 <= tree[p * 2].rf)  change(p * 2, l, mid, line);
    else if(line.y1 >= tree[p * 2 + 1].lf)  change(p * 2 + 1, mid, r, line);
    else{
        ty tmp = line;
        tmp.y2 = tree[p * 2].rf;
        change(p * 2, l, mid, tmp);
        tmp = line;
        tmp.y1 = tree[p * 2 + 1].lf;
        change(p * 2 + 1, mid, r, tmp);
    }
    calc(p, l, r);
```

```
void calc(int p, int l, int r)
{
    if(tree[p].c >= 2)
    {

        tree[p].cnt2 = tree[p].rf - tree[p].lf;
        return;
    }
    else if(tree[p].c == 1)
    {
        tree[p].cnt = tree[p].rf - tree[p].lf;
        if(l + 1 == r)  tree[p].cnt2 = 0;
        else  tree[p].cnt2 = tree[p * 2].cnt + tree[p * 2  + 1].cnt;
    }
     else
    {
        if(l + 1 == r)  tree[p].cnt = tree[p].cnt2 = 0;
        else
        {
            tree[p].cnt = tree[p * 2].cnt + tree[p * 2  + 1].cnt;
            tree[p].cnt2 = tree[p * 2].cnt2 + tree[p * 2  + 1].cnt2;
        }
    }


}
```

树状数组

- ——树状数组能做的题线段树都能，但线段树能做的题树状数组不一定能……
- 那为啥要学?
- 代码短啊，时空常数小啊!


[1,8]
[1,4]
[5,8]
[1,2]
[3,4]
[5,6]
[7,8]
1
2
3
4
5
6
7
8


C[8]
C[4]
C[2]
C[6]
C[1]
C[3]
C[5]
C[7]
A[1]
A[2]
A[3]
A[4]
A[5]
A[6]
A[7]
A[8]
......

$C1 = A1 \qquad C2 = A1 + A2$

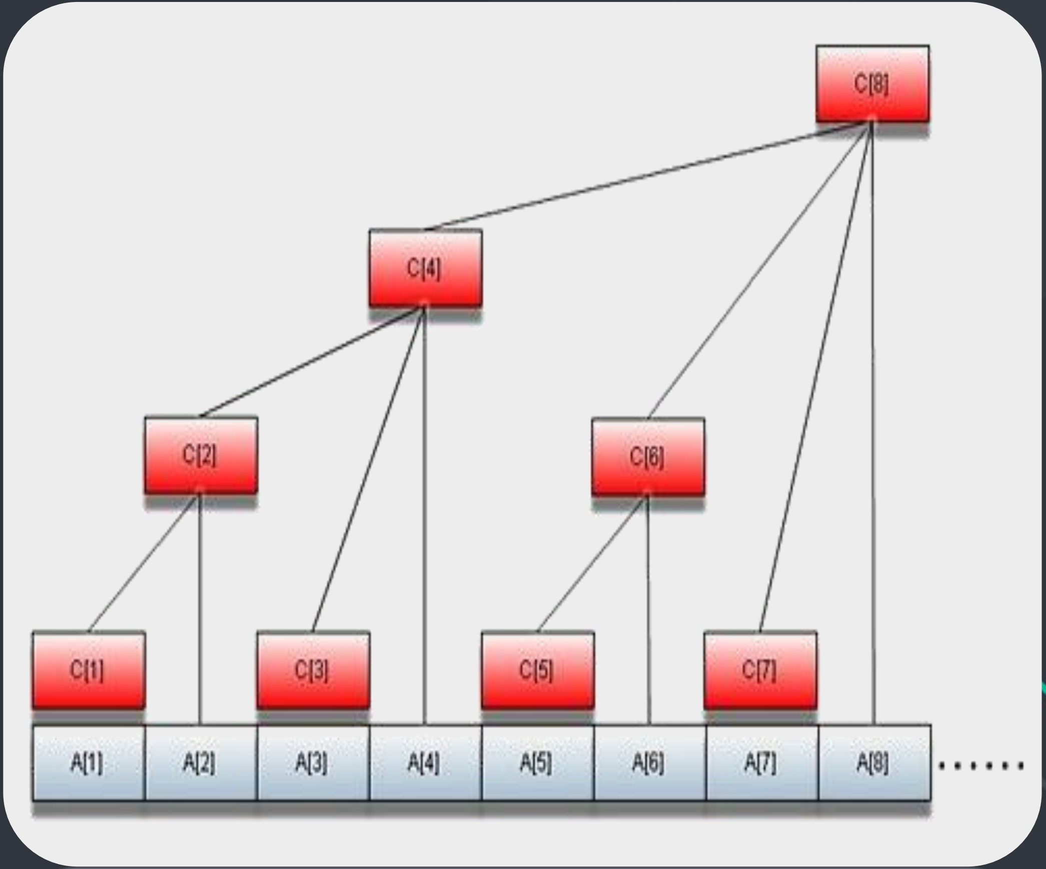
$C3 = A3 \qquad C4 = A1 + A2 + A3 + A4$

$C5 = A5 \qquad C6 = A5 + A6$

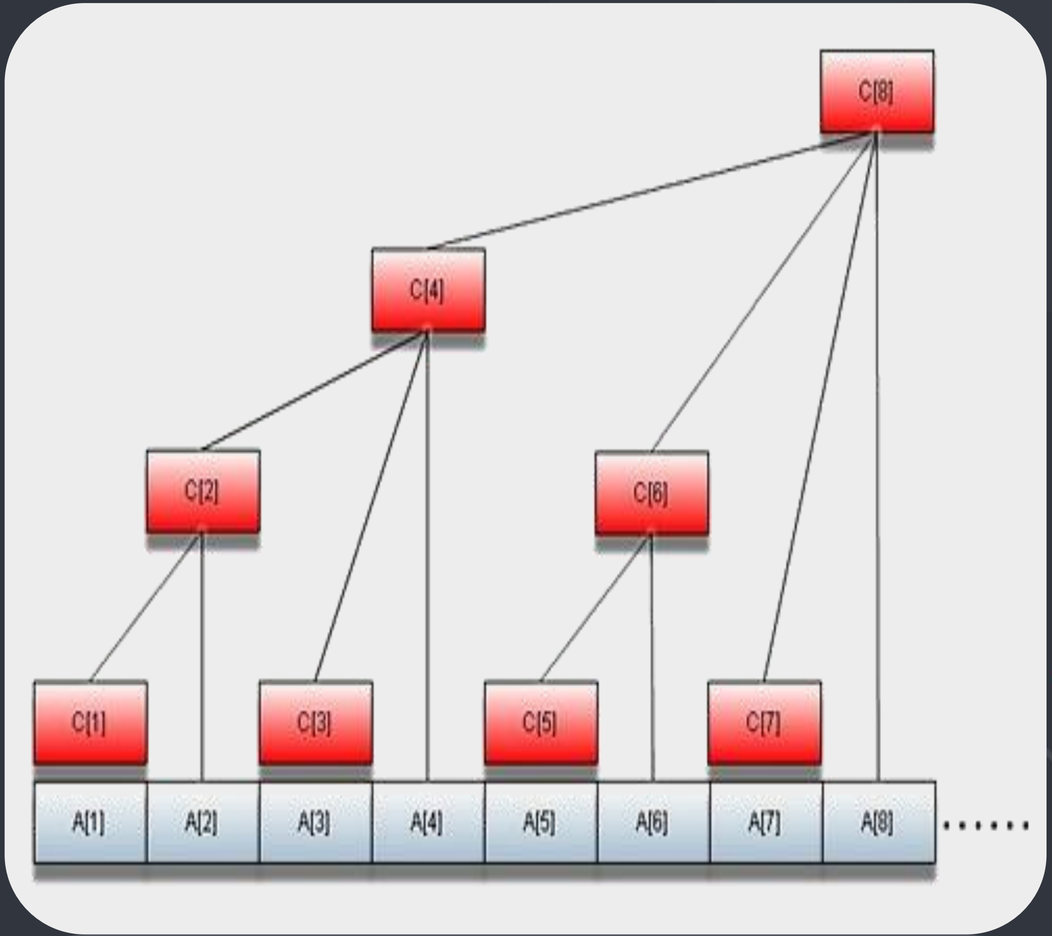
$C7 = A7 \qquad C8 = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8$

...

$C16 = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8 + A9 + A10 + A11 + A12 + A13 + A14 + A15 + A16$



- 这里有一个有趣的性质：
- 设节点编号为x，那么这个节点管辖的区间为2^k（其中k为x二进制末尾0的个数）个元素。因为这个区间最后一个元素必然为Ax，
- 所以很明显：Cn = A(n – 2^k + 1) + ... + An
- 算这个2^k有一个快捷的办法，定义一个函数如下即可：
- int lowbit(int x){ return x&(-x); }



```
int lowbit(int x)
{
    return x&(-x);
}

void add(int i,int val)
{
    while(i<=n)
    {
        c[i]+=val;
        i+=lowbit(i);
    }
}
```

```
int sum(int i)
{
    int s=0;
    while(i>0)
    {
        s+=c[i];
        i-=lowbit(i);
    }
    return s;
}
```

例1：

- 给定一个长度为N的序列A，所有元素初值为0。接下来有M次操作或询问：
- 操作：输入格式：1 D K，对于所有满足1≤i≤且i≡0(mod　D) 的i，将Ai加上K。
- 询问：输入格式：2 L R，询问区间和。

- D比较大的时候——直接单点修改——改的点数较小
- D小的时候怎么办呢?
- 开个数组 add[i] 表示D=i 的数都加上了几
- 算区间和的时候 访问每个add[i] 算其对区间和的贡献即可