线段树
—— 高级数据结构之一

北京大学信息学院

《数据结构与算法实习》

张路

2017.9

大纲

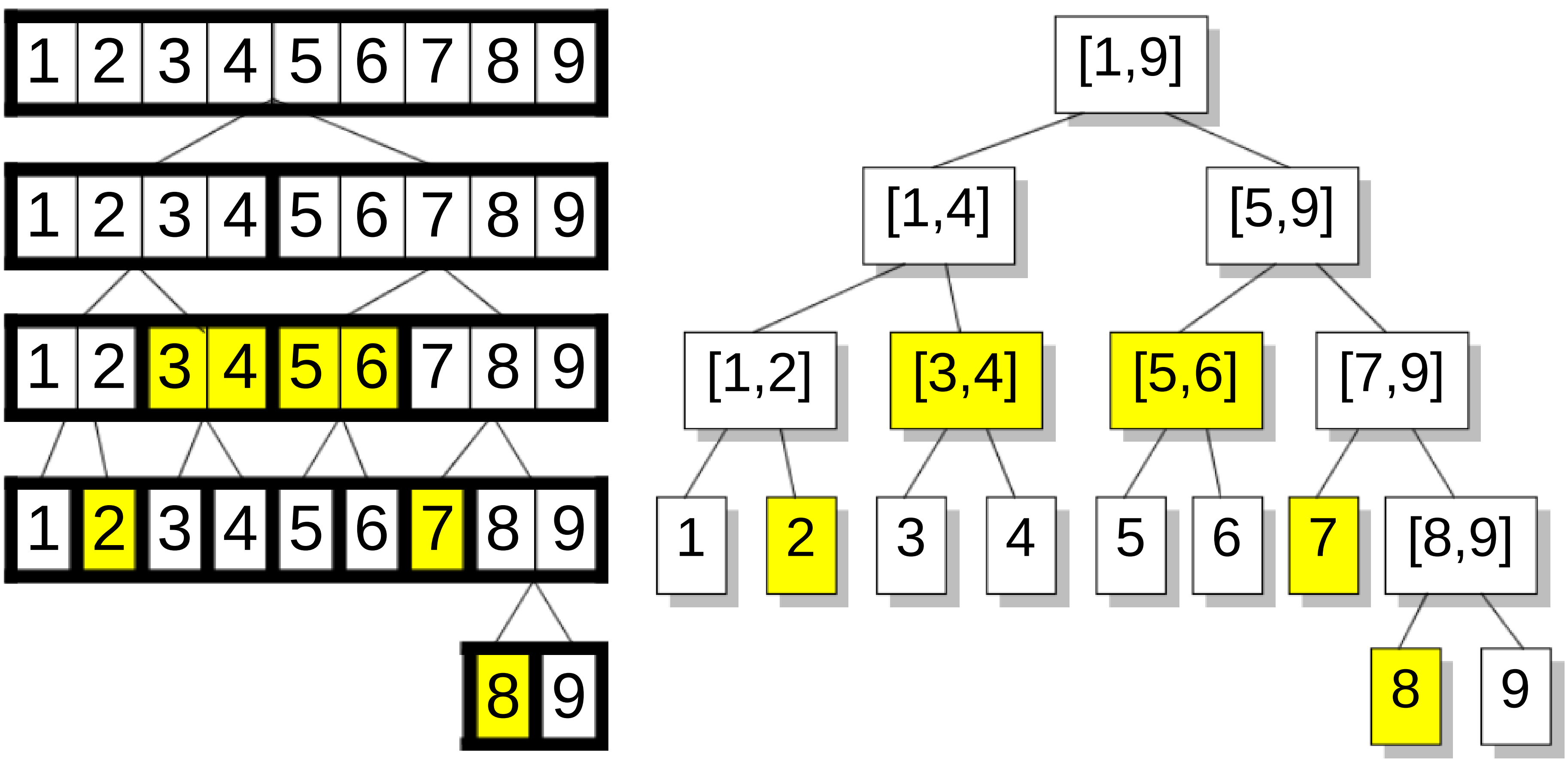
- 1. 线段树的定义
- 2. RMQ
- 3. 染色问题
- 4. 例题
- 5. 推广

- 线段树的定义
  - 线段树是一棵二叉树，树上每个结点都表示一个线段（区间）。
- 线段树的构造
  - 每一个非叶子结点 [a,b]，它的左儿子和右儿子分别是这条线段的左半段和右半段

```
                [a,                    b]
                     /            \
          [a,(a+b)/2]      [(a+b)/2+1,b]
```

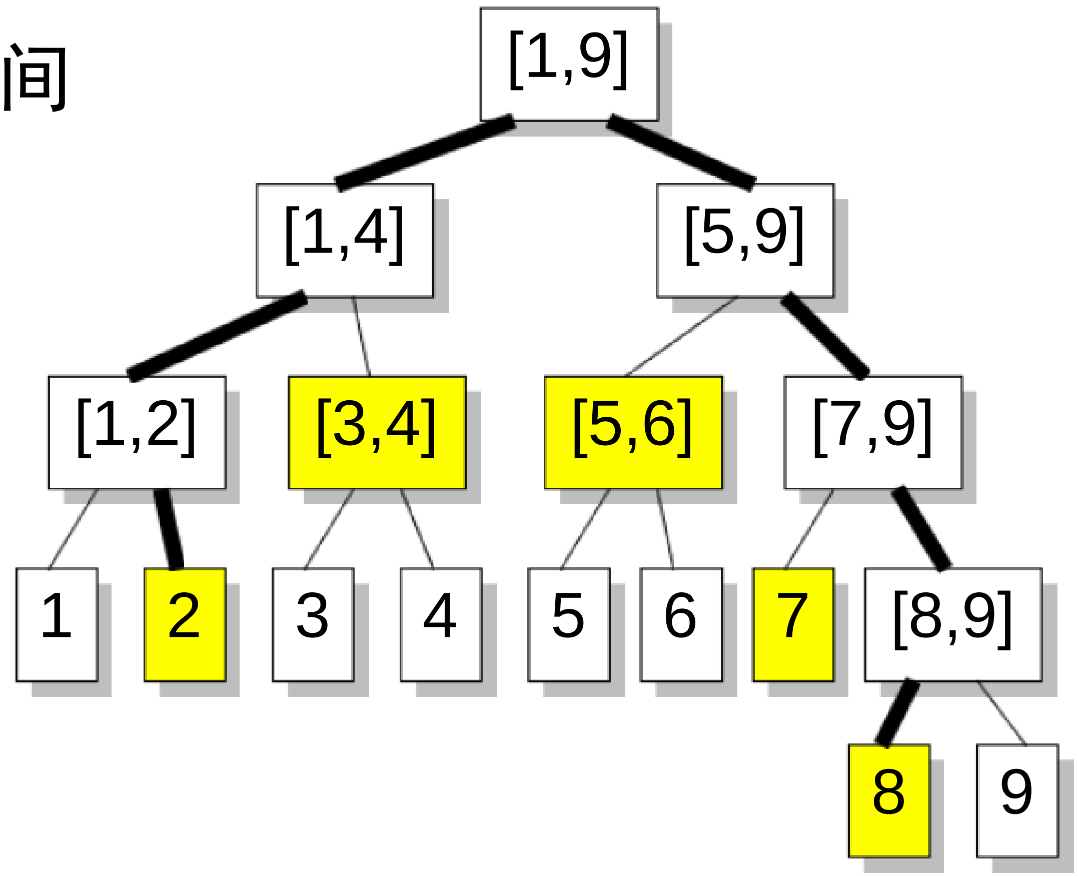
线段树的定义

- 线段[1, 9]的线段树和对区间[2, 8]的分解

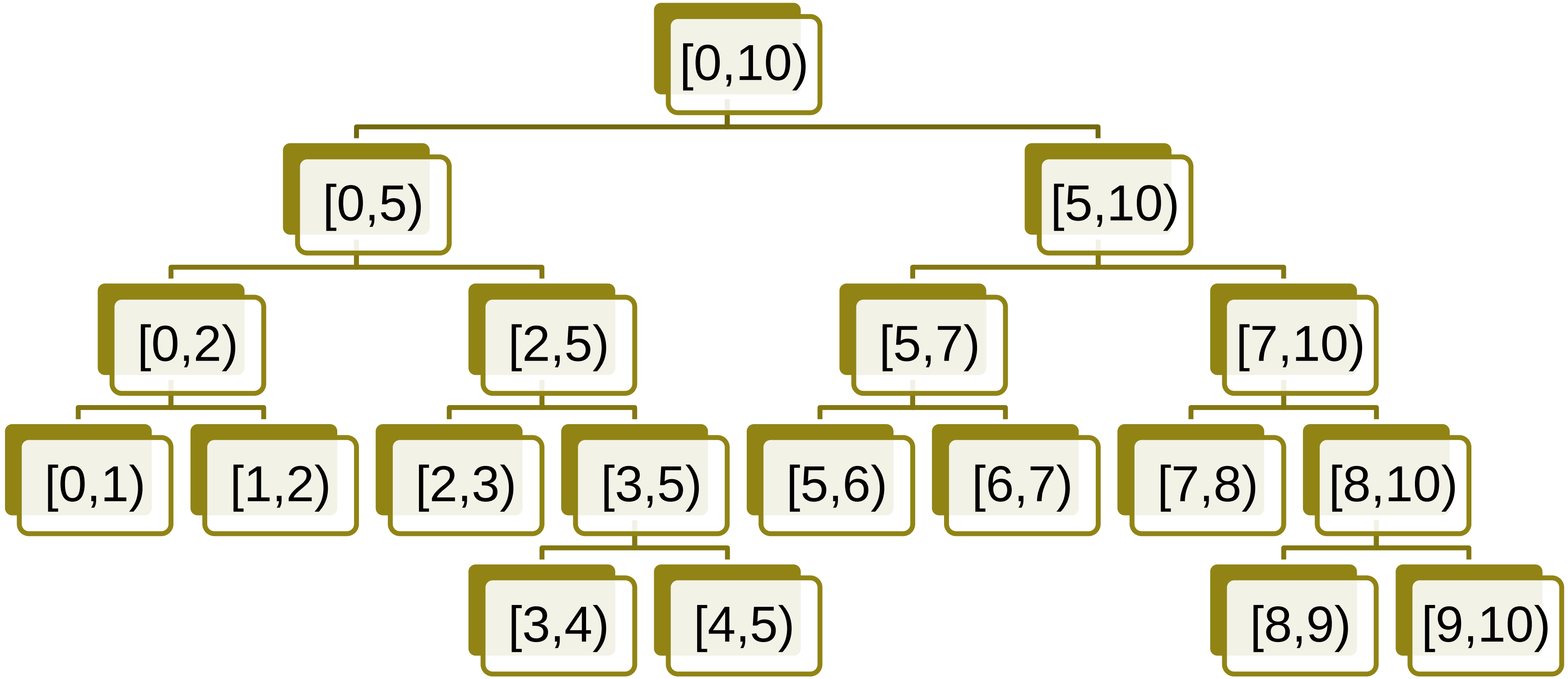


基本算法

- **找点:** 根据定义，从根一直走到叶子log n层
- **区间分解:**
  - 每层最多两个区间
  - 总时间O(log n)

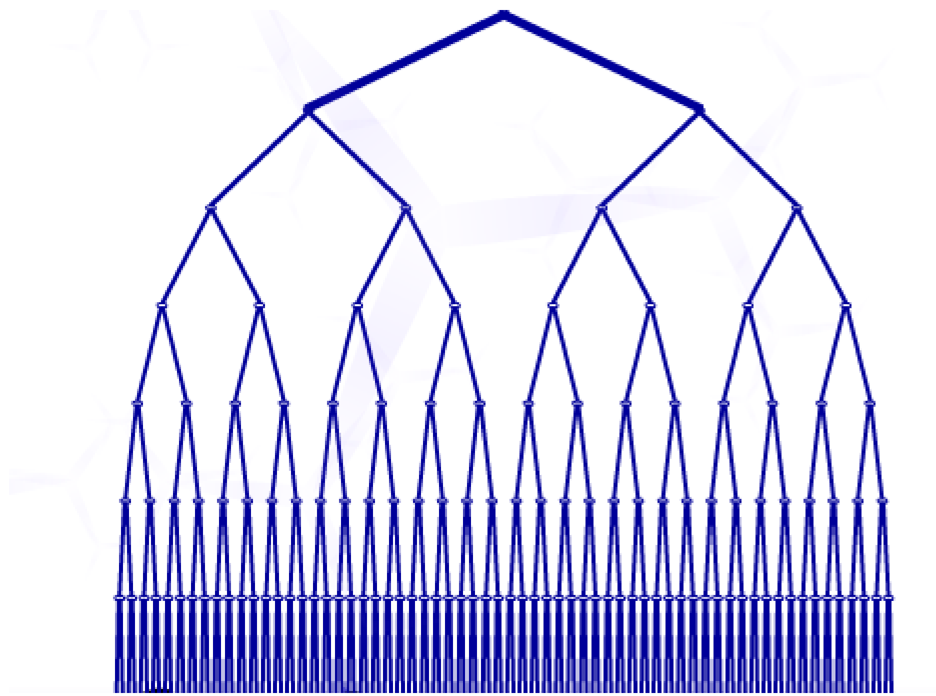


另一种开区间的表示方法

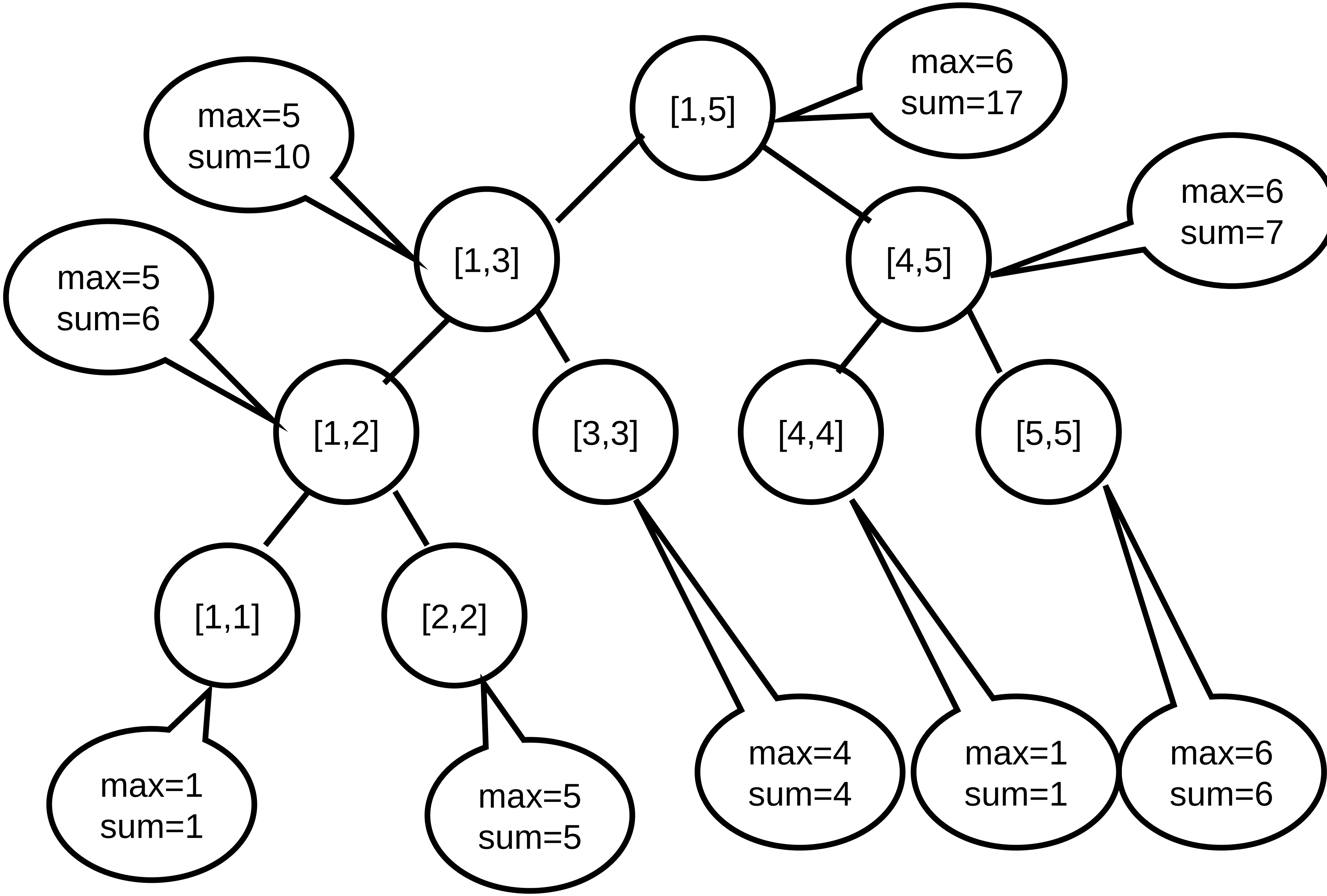


线段树的定义

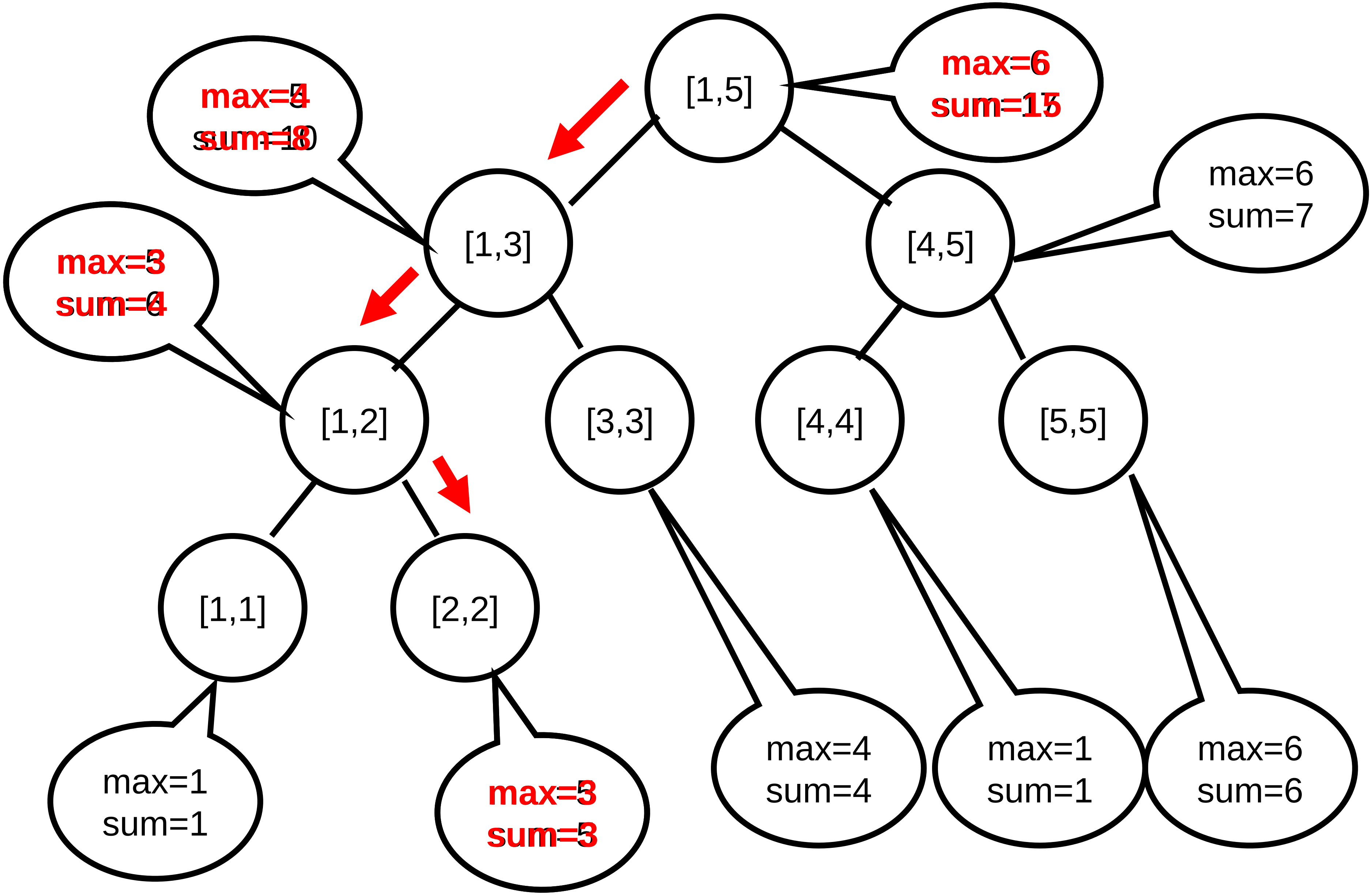
N= 128



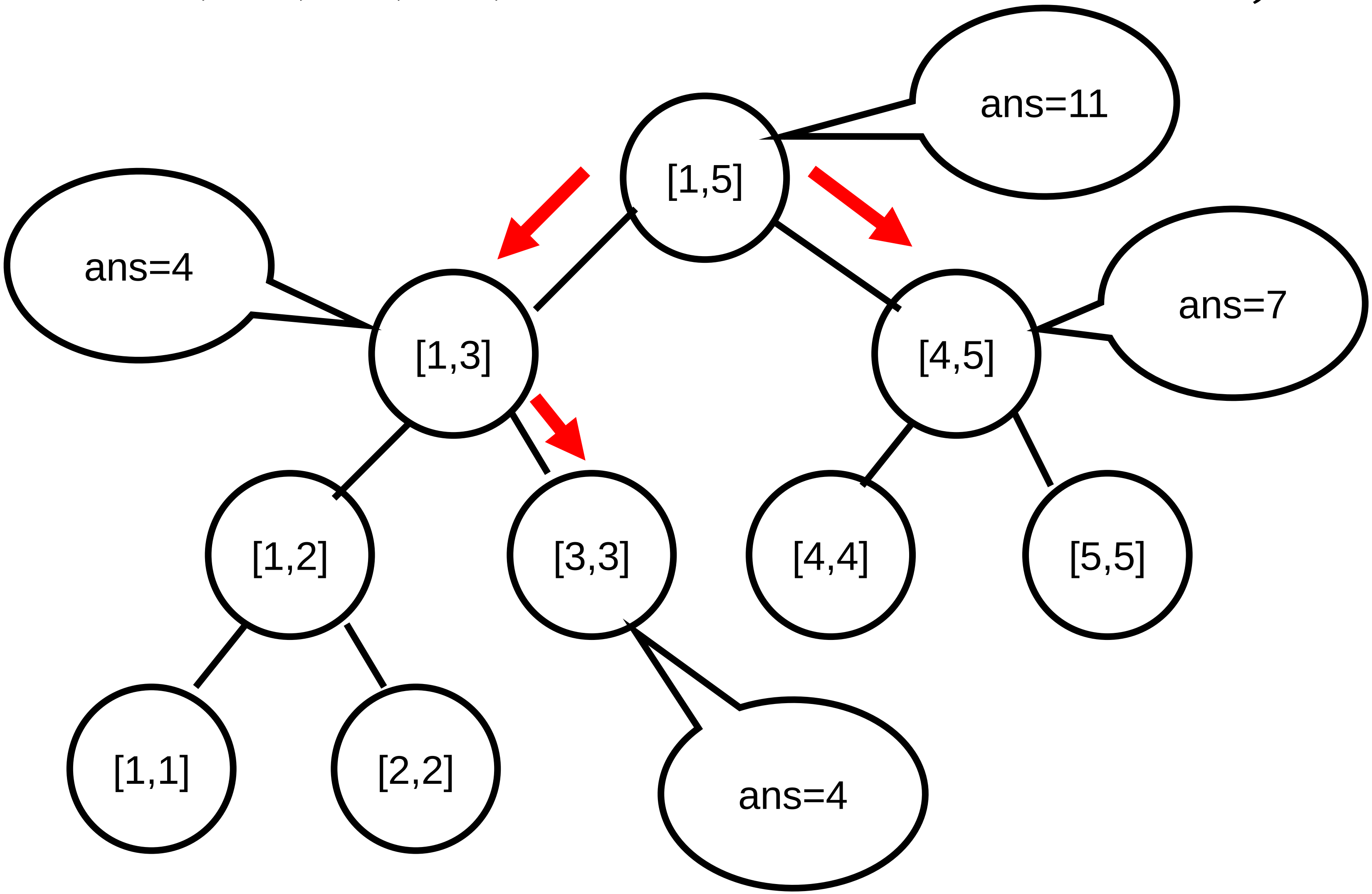
数组[1, 5, 4, 1, 6]


[1,5]
max=6
sum=17
max=5
sum=10
[1,3]
[4,5]
max=6
sum=7
max=5
sum=6
[1,2]
[3,3]
[4,4]
[5,5]
[1,1]
[2,2]
max=1
sum=1
max=5
sum=5
max=4
sum=4
max=1
sum=1
max=6
sum=6

[1, 5, 4, 1, 6]　更新，a[2]=3


[1,5]
max=6
sum=15
[1,3]
max=4
sum=8
[4,5]
max=6
sum=7
[1,2]
max=3
sum=4
[3,3]
[4,4]
[5,5]
[1,1]
[2,2]
max=1
sum=1
max=3
sum=3
max=4
sum=4
max=1
sum=1
max=6
sum=6

数组[1,5,4,1,6]　　查询 sum(3,5)?



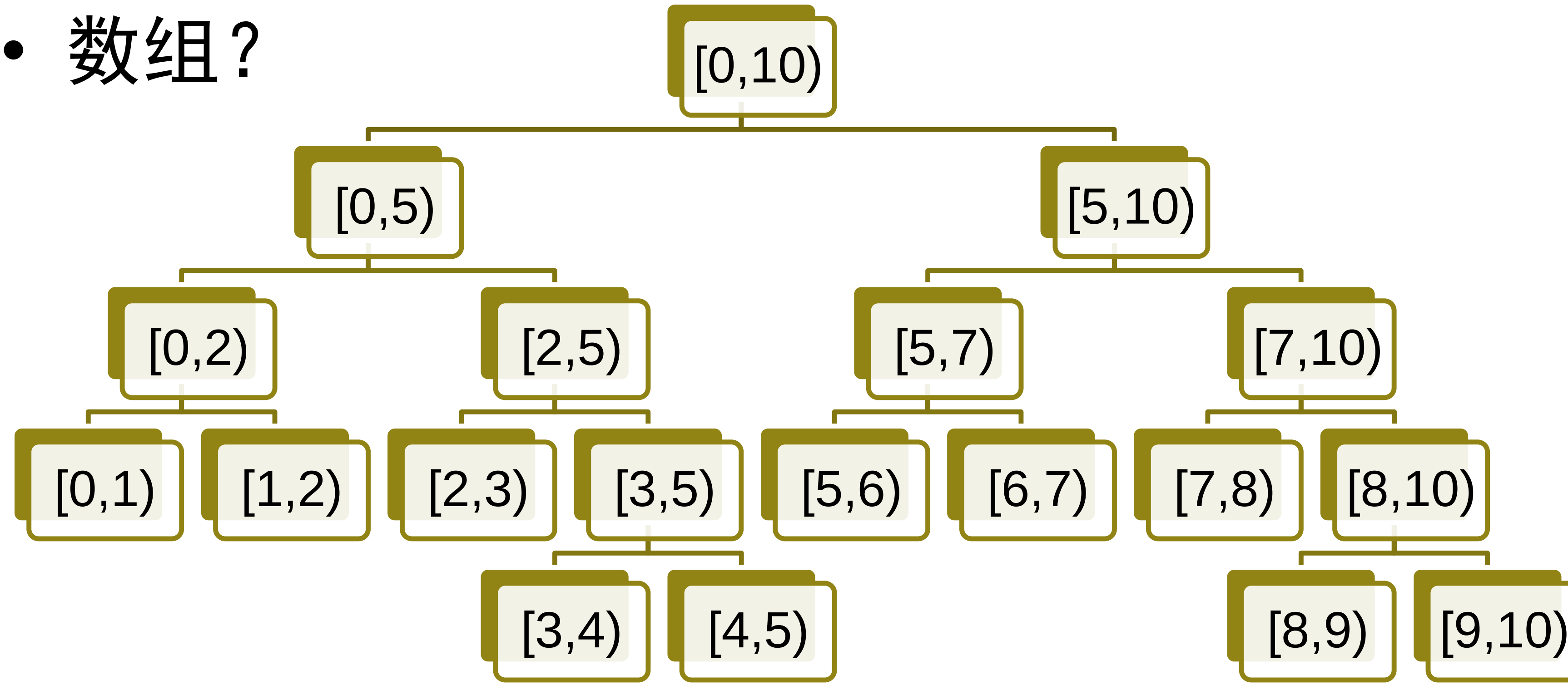
线段树的特征

- 1、线段树的深度不超过$\log_2(n)+1$（向上取整，n是根结点对应区间的长度）。
- 2、线段树上，任意一个区间被分解后得到的“终止结点”数目都是log(n)量级。

- 这些结论为线段树能在O(log(n))的时间内完成一个区间的建树，插入数据，查找、统计等工作，提供了理论依据

怎么存？

- 链式？
- 数组？



```
• struct Node
• {
•         int         a, b;                    //a代表左界，b代表右界
•         Node        *lch, *rch;              //lch代表左儿子，rch代表右儿子
• } st[maxn * 2];
• int m = 0;

• void build (int i, int x, int y)
• {
•         st[i].a = x;        st[i].b = y;
•         if (x < y)
•         {
•                 st[i].lch = &st[++m];        build(m,   x,          (x+y)/2);
•                 st[i].rch = &st[++m];        build(m,   (x+y)/2+1,        y);
•         }
• }
```

2. RMQ问题

– 长度为n的数列，有两个操作

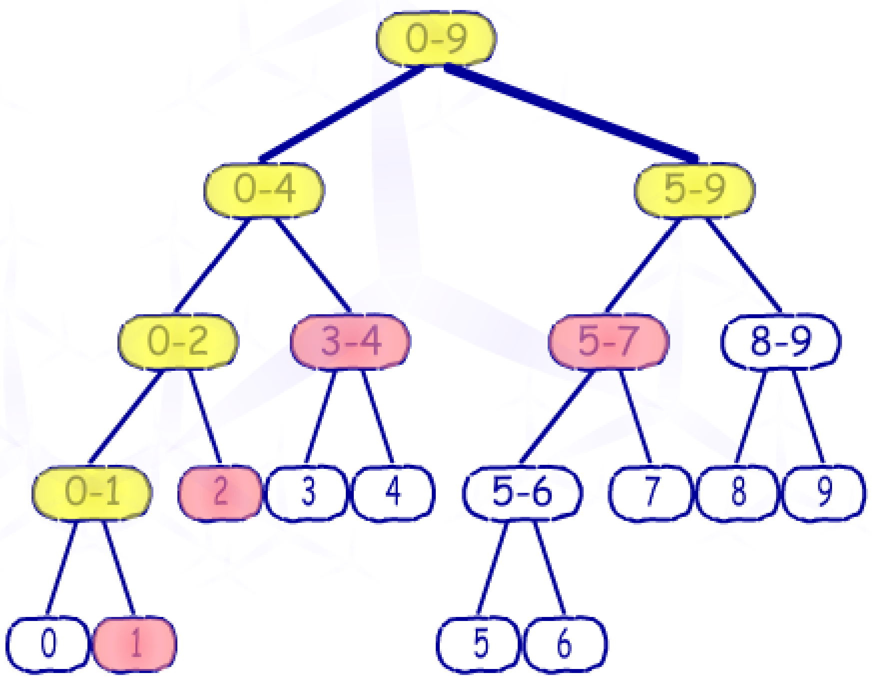
（1）将位置k上的值修改为x；

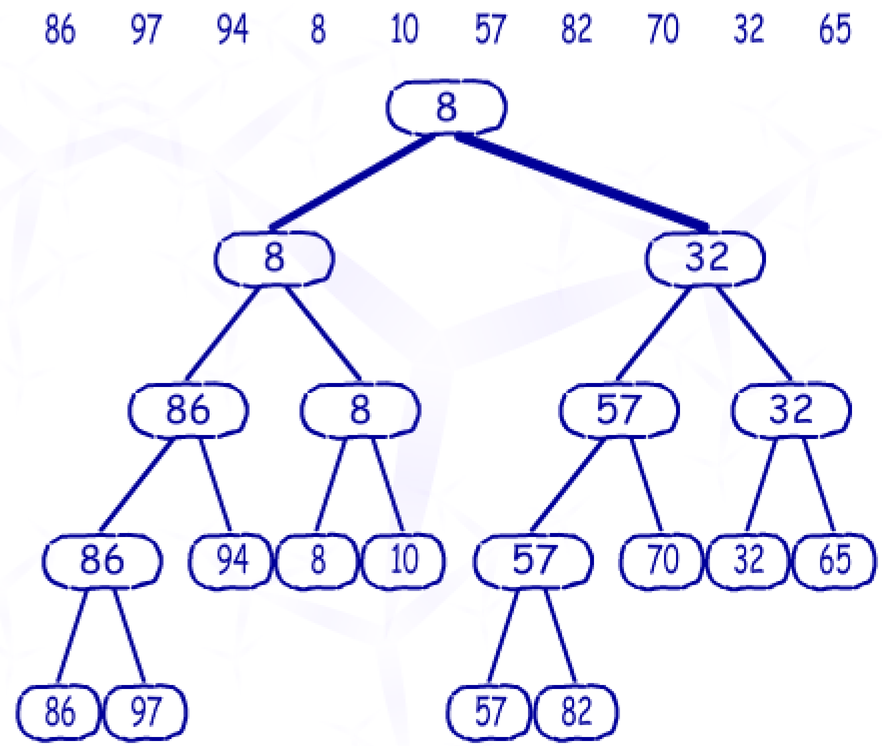
（2）询问区间[l，r]上的最小值。

朴素方法？

- 修改O(1)
- 查询O(n)

线段树的建立


0-9
0-4
5-9
0-2
3-4
5-7
8-9
0-1
2
3
4
5-6
7
8
9
0
1
5
6


86 97 94 8 10 57 82 70 32 65
8
8
32
86
8
57
32
86
94
8
10
57
70
32
65
86
97
57
82

线段树的构建

- function 以结点v为根建树、区间为[l,r]
- {
-     对结点v初始化
-     if (l!=r)
-     {
-         以v的左孩子为根建树、区间为[l,(l+r)/2]
-         以v的右孩子为根建树、区间为[(l+r)/2+1,r]
-     }
- }

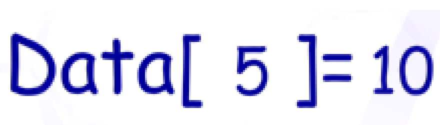
建树过程:

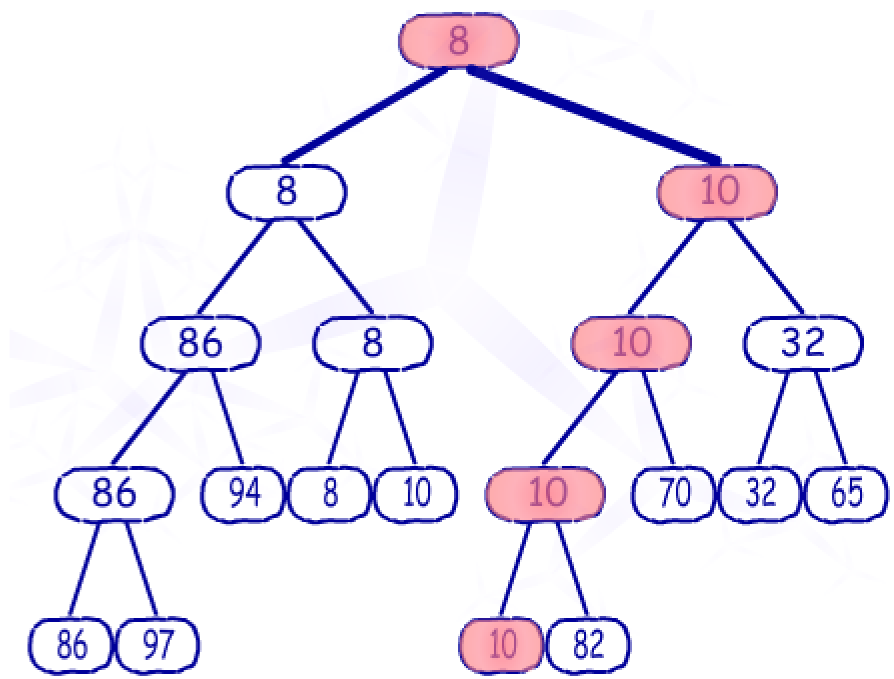
```
if (x < y)
{

        st[i].lch = &st[++m];        build(m,   x,           (x+y)/2);
        st[i].rch = &st[++m];        build(m,   (x+y)/2+1,        y);
        st[i].f = min(st[i].lch->f, st[i].rch->f);
        //结点的最小值等于左右儿子的最小值较小者
}
else st[i].f = a[x];    //叶子结点最小值就是当前位置的值
```

线段树的修改

- 把data[5]的值修改为10





线段树的修改

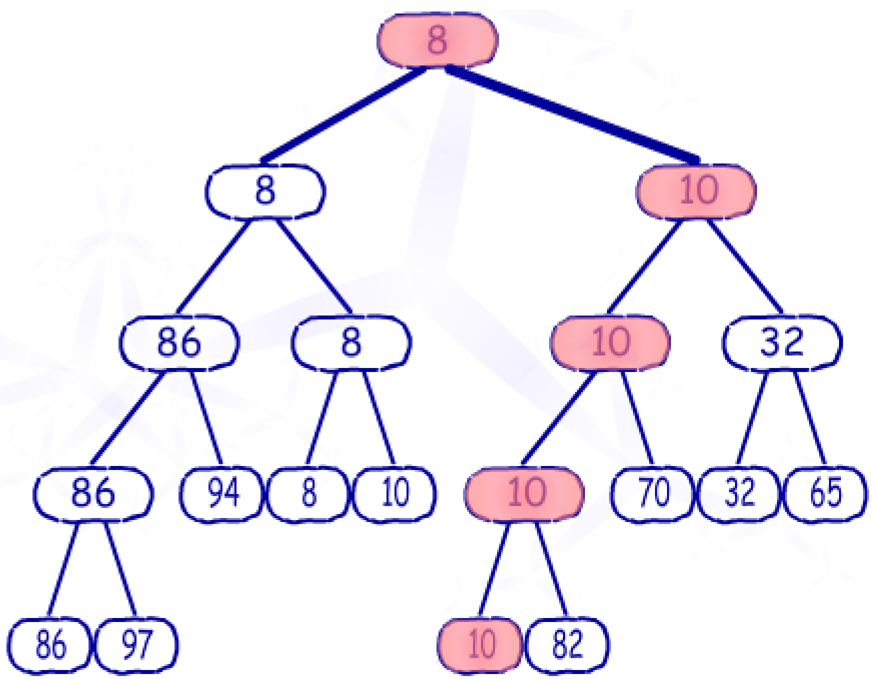
- function 修改以结点v为根的子树、区间为[l,r]
- {
-     如果修改点不属于[l,r] 跳出
-     if (l!=r)
-     {
-         以v的左孩子为根修改树、区间为[l,(l+r)/2]
-         以v的右孩子为根修改树、区间为[(l+r)/2+1,r]
-     }
-     修改结点v的值
- }

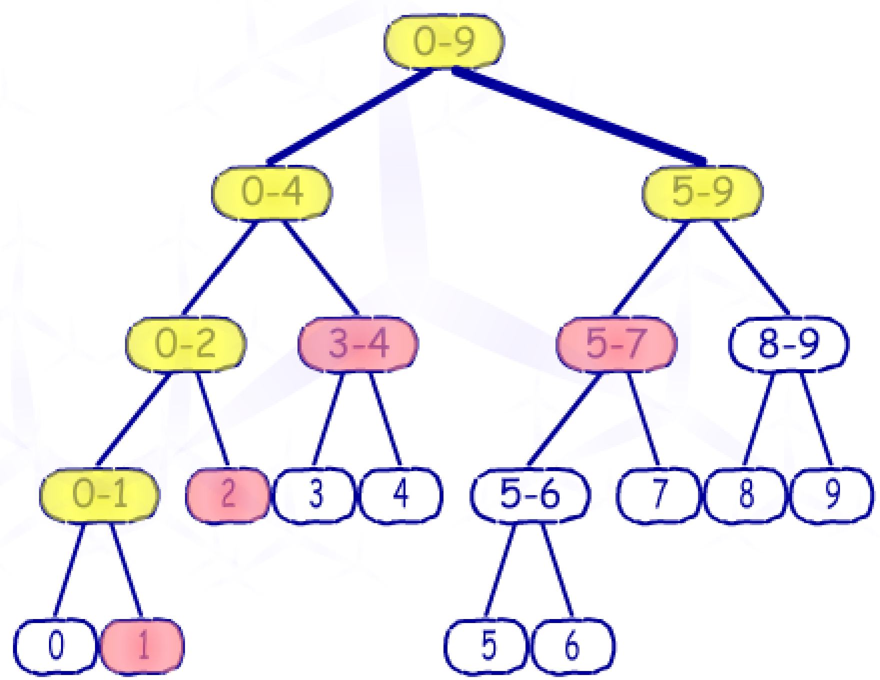
修改操作：

```
• void change(Node *p, int k, int x)        //将k位置上的数修改为x
• {
•         if (p->a != p->b)
•         {
•                 int mid = (p->a + p->b) / 2;
•                 if (k <= mid) change(p->lch, k, x);   //k位置属于左半段
•                 if (k > mid) change(p->rch, k, x);    //k位置属于右半段
•                 p->f = min(p->lch->f, p->rch->f);    //更新最小值
•         }
•         else p->f = x;                      //叶子结点最小值等于x
• }
```

线段树的查询

- 查询data[1]...data[7]中的最小值
- 询问区间[1，7]，下图被染色的结点为遍历过的结点，红色结点更新了答案。





线段树的查询

- function 在结点v查询区间[x,y]
- {
-     if v所表示的区间与[x,y]的交集为空， 跳出
-     if v所表示的区间完全属于[x,y] 选取v
-     else
-     {
-         在v的左孩子查询[x,y]
-         在v的右孩子查询[x,y]
-     }
- }

查询操作：

```
• void query(Node *p, int x, int y)
• {
•     if (p->a >= x && p->b <= y) ans = min(ans, p->f);
•     //如果当前线段[a,b]完全包含于询问区间[x,y]，更新答案
•     else
•     {
•         int mid = (p->a + p->b) / 2;
•         if (x <= mid) query(p->lch, x, y);
          //左半段与询问区间有相交部分，继续递归
•         if (y > mid) query(p->rch, x, y);
          //右半段与询问区间有相交部分，继续递归
•     }
• }
```

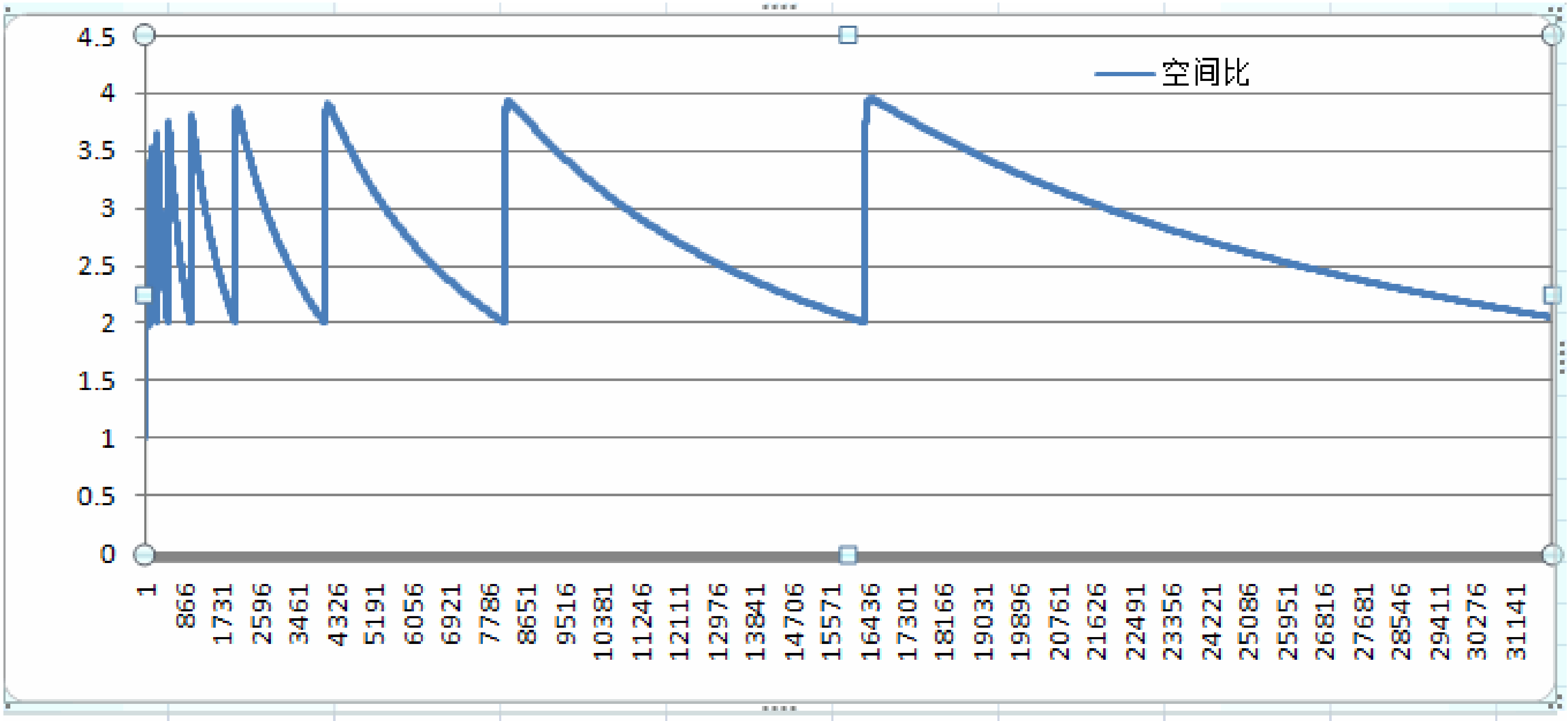
- 复杂度
  - 每次修改的复杂度为O(log n)
  - 每次询问的复杂度为O(log n)
    - 选出的区间相连后正好覆盖询问区间，同一层被访问的结点不会超过4个（为什么呢O_O自己模拟一下就知道原因了）。
  - 总复杂度为O(m log n)，m为操作数

线段树——空间复杂度

- 设长度为n的数组在线段树中有F(n)个结点
  - F(n) 包括没有使用的下标为0的结点
  - 若$n=2^k$, $F(n)=2^{(k+1)}$
  - 若$n=2^{(k+1)}$，$F(n)=2^{(k+2)}$

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

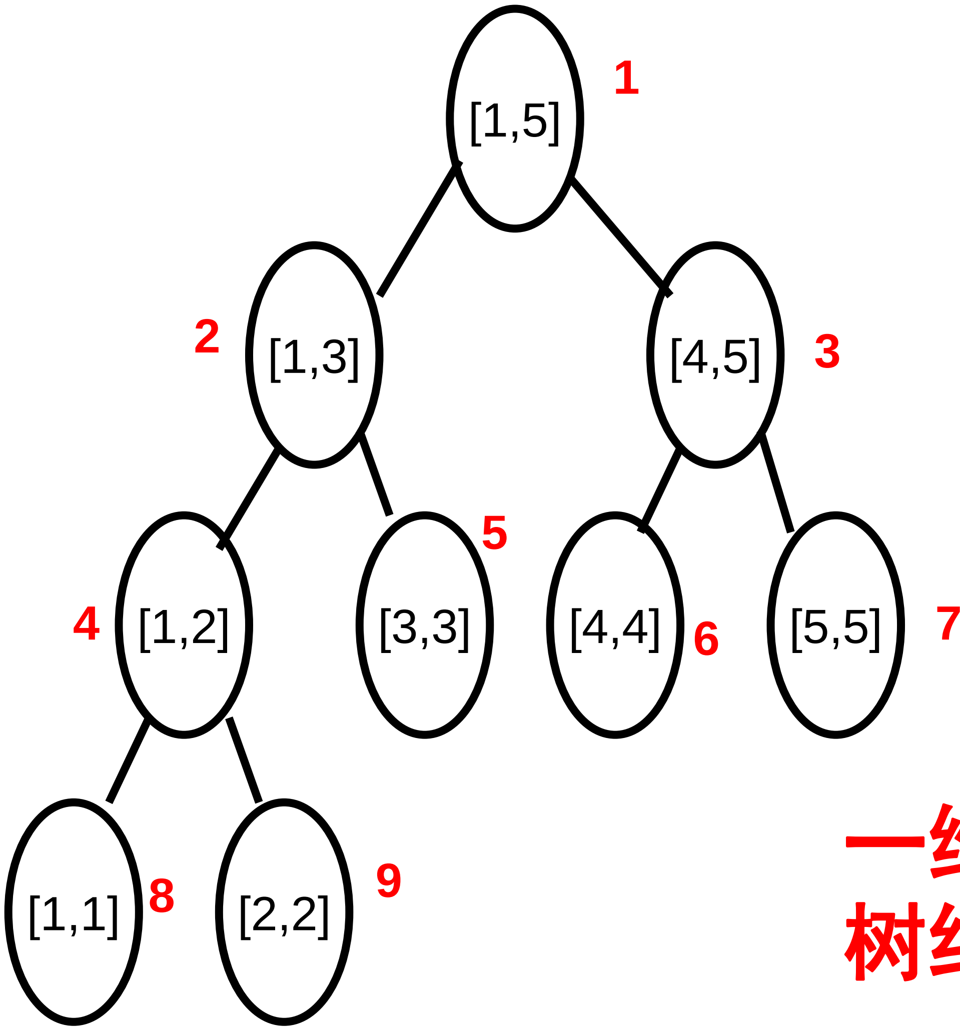
线段树——空间复杂度



（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——数组实现



我们用一个数组a记录结点，且根结点的下标为1，

对于任一结点a[k]，
它的左儿子为a[2*k]
它的右儿子为a[2*k+1]

一维数组即实现了线段树结点的保存

线段树——代码实现(建树)

- 建立一棵线段树，并记录原数组信息

```
void build(int id,int l,int r)
{
     tree[id].left=l; tree[id].right=r;
     if (l==r)
     {
             tree[id].sum=tree[id].max=a[l];
     }
     else
     {
             int mid=(l+r)/2;
             build(id*2,l,mid);
             build(id*2+1,mid+1,r);
             tree[id].sum=tree[id*2].sum+tree[id*2+1].sum;
             tree[id].max=max(tree[id*2].max,tree[id*2+1].max);
     }
}
```

如果原数组从**a[1]~a[n]**,调用**build(1,1,n)**即可

线段树——代码实现(更新)

- 更新某个点的数值，并维护相关点的信息

```
void update(int id,int pos,int val)
{
     if (tree[id].left==tree[id].right)
     {
             tree[id].sum=tree[id].max=val;
     }
     else
     {
             int mid=(tree[id].left+tree[id].right)/2;
             if (pos<=mid) update(id*2,pos,val);
             else update(id*2+1,pos,val);
             tree[id].sum=tree[id*2].sum+tree[id*2+1].sum;
             tree[id].max=max(tree[id*2].max,tree[id*2+1].max)
     }
}
```

若更新**a[k]**的值为**val**，调用**update(1,k,val)**即可

线段树——代码实现(查询)

- 查询某区间内元素的和或最大值(以总和为例)

```
void query(int id,int l,int r)
{
    if (tree[id].left==l&&tree[id].right==r)
          return tree[id].sum; //询问总和
    else
    {
          int mid=(tree[id].left+tree[id].right)/2;
          if (r<=mid) return query(id*2,l,r);
          else if (l>mid) return query(id*2+1,l,r)
          else
             return query(id*2,l,mid)+query(id*2+1,mid+1,r);
    }
}
```

调用**query(1,l,r)**即可查询**[l,r]**区间内元素的总和

- 3. 染色问题
  - 在数轴上有两类操作

    （1）将区间[a,b]涂上颜色，或者擦去颜色；

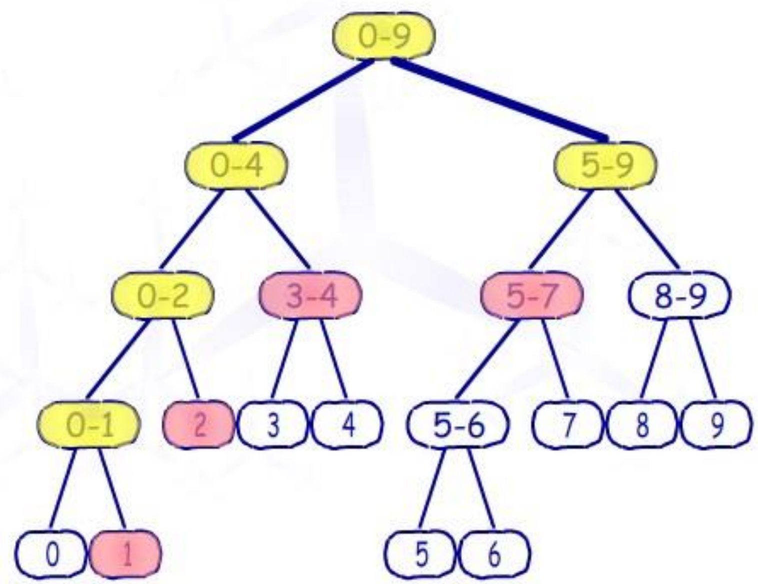
    （2）询问当前有多少条单位线段[k,k+1]被涂上了颜色。

- 初步想法：每个结点记录当前线段总共包含多少个被染色的单位线段，每次染色将所有被影响的结点更新。

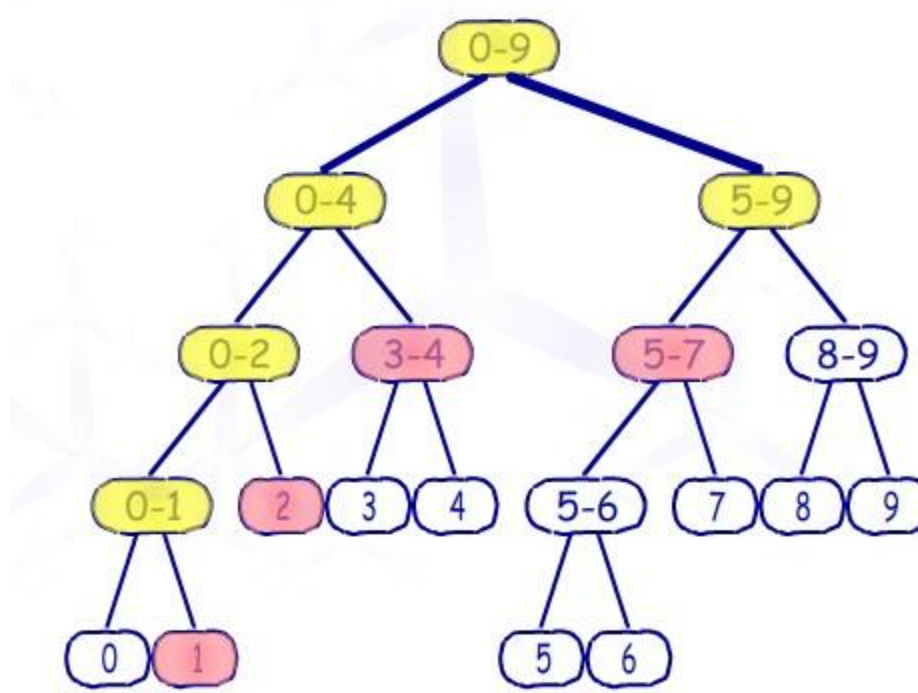
- 但是这种做法每次修改操作要更新所有被影响的结点，若染色[l，r]区间，则被影响的结点一共有(r-l)*2个，已经不是O(log n)级别的了，与朴素的暴力解法差不多，何必再写线段树呢。

- 解决方法：由于修改操作也是和询问一样，是对一段区间进行的，我们也像询问那样只访问到刚好被包含的区间线段，不再向下遍历。

修改[1，7]→



- 访问至红色结点，给它加个标记，表示当前结点以及它的子树都被染色过，直到下次要从这个结点往下走，再把染色标记带下去。
- 这种标记思想的精髓是红色结点以下的信息暂时不更新，但红色结点以上的信息是正确的。



- st[i].flag表示标记
  - flag == 0表示当前结点没有标记，也就是既没染色也没有擦除颜色。
  - flag == 1表示当前结点以及整棵子树都被染色。
  - flag == -1表示当前结点以及整棵子树都被擦除颜色。

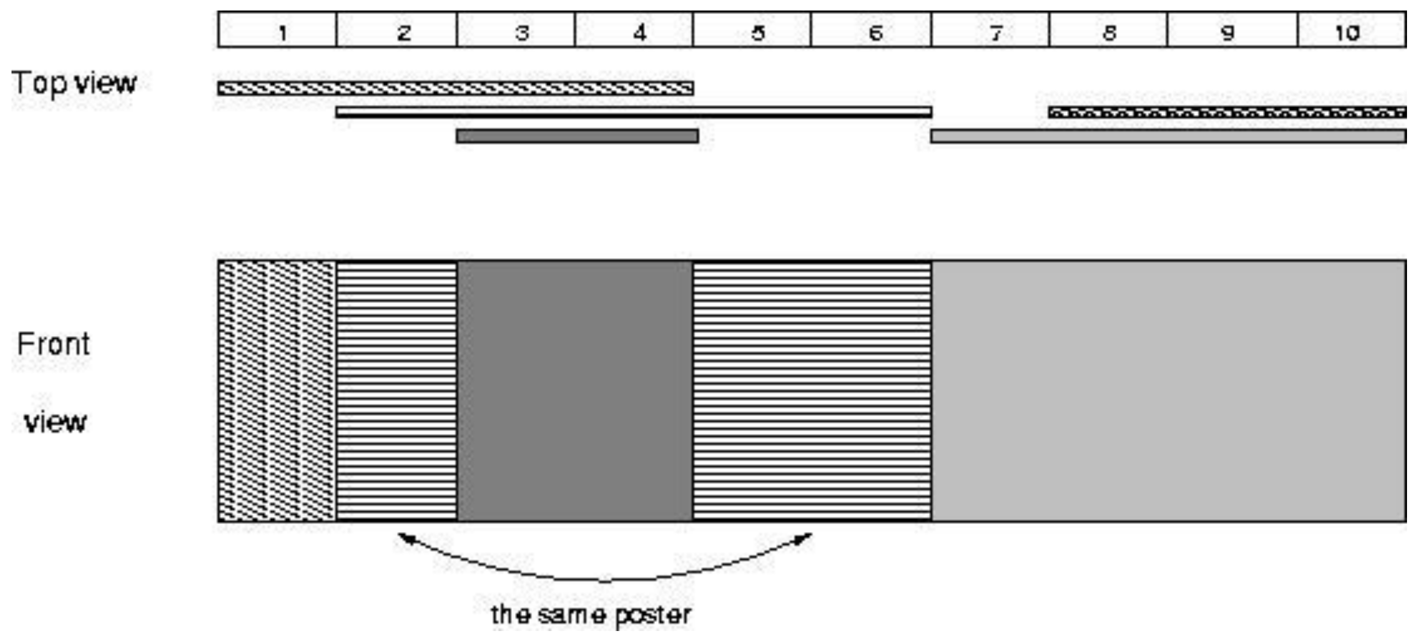
```
void change(Node *p, int x, int y, int k)        //将区间[x,y]染为k，k==1表示染色，k==0表示擦除
{
        if (p->a >= x && p->b <= y)                      //当前线段被包含于修改区间
        {
                p->flag = k;                  //更改标记
                {           根据标记整理当前结点信息        }
        }
        else
        {
                if (p->flag != 0)             //如果当前结点有标记
                {
                        p->lch->flag = p->rch->flag = p->flag;           //将标记传给两个孩子
                        {           根据标记整理两个孩子的信息     }
                        p->flag = 0;                          //自己清除标记
                }
                int mid = (p->a + p->b) / 2;
                if (x <= mid) change(p->lch, x, y, k);
                if (y > mid) change(p->rch, x, y, k);
                {           根据两个孩子的信息更新当前结点              }
        }
}
```

- 询问操作类似，也是再访问两个孩子结点前先将标记传下去。
- 这样我们两项操作的复杂度还是O(log n)了

- 标记思想不仅在线段树中很常见，平衡树中也非常有用。

4. 例题 POJ 2528 Mayor's posters

给定一些海报，可能互相重叠，告诉你每个海报宽度（高度都一样）和先后叠放次序，问没有被完全盖住的海报有多少张。



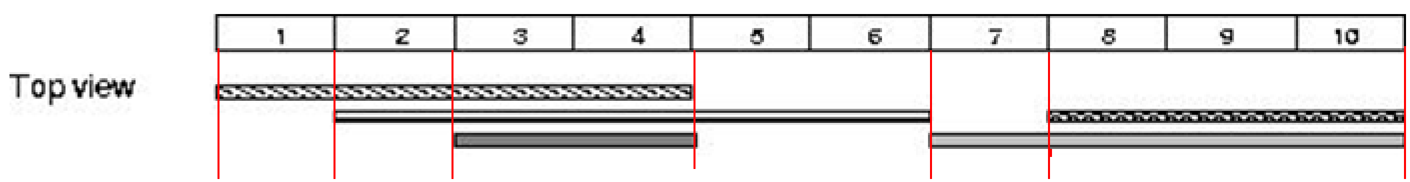
海报最多10,000张，但是墙有10,000,000块瓷砖长。海报端点不会落在瓷砖中间。

POJ 2528 Mayor's posters

如果每个叶子结点都代表一块瓷砖，那么线段树会导致MLE，即单位区间的数目太多。

实际上，由于最多10,000个海报，共计20,000个端点，这些端点把墙最多分成19,999个区间（题意为整个墙都会被盖到）

我们只要对这19,999个区间编号，然后建树即可。这就是离散化。



POJ 2528 Mayor's posters

如果海报端点坐标是浮点数，其实也一样处理。

树结点要保存哪些信息，而且这些信息该如何动态更新呢？

POJ 2528 Mayor's posters

```
struct CNode
{
        int L,R;
        bool bCovered;
         CNode * pLeft, * pRight;
};
```

bCovered表示本区间是否已经完全被海报盖住

关键： 插入数据的顺序 ------ 从上往下依次插入每张海报

例2

- 点在线段上出现次数的查询问题

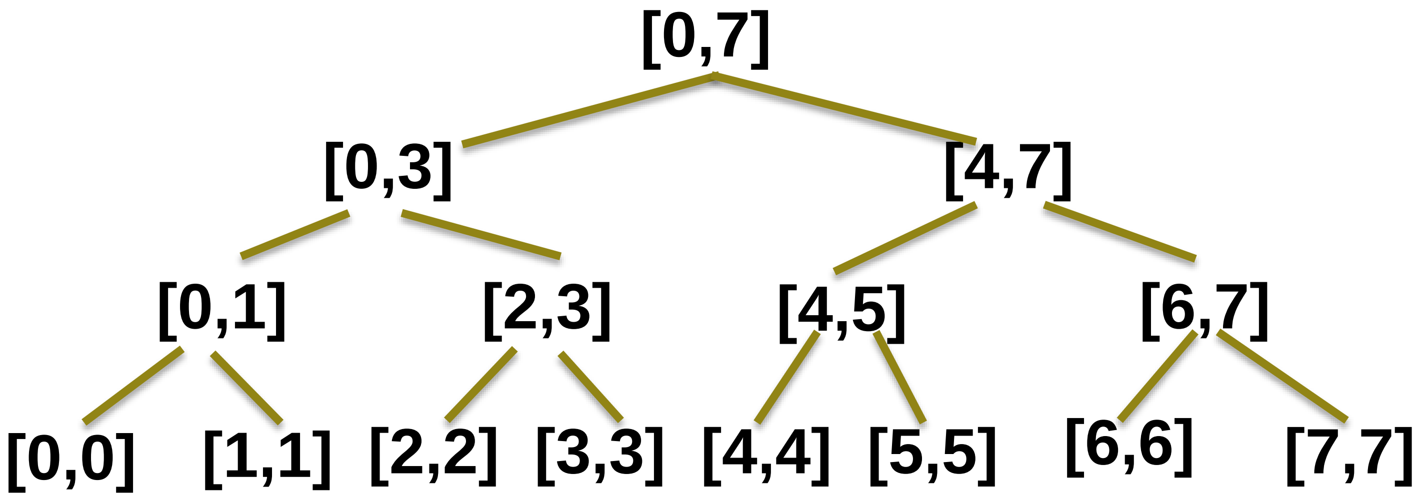
在不大于N=30000的自然数范围内，已知有b条线段，且有m个查询点（m,b<30000)，问：这些点在多少条线段上出现过？

分析：

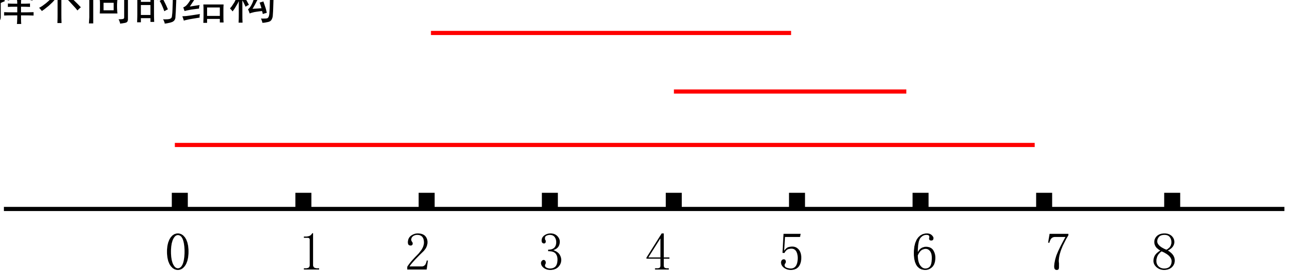
- 思路一：每读入一个点，在所有线段比较一下，看是否在线段中。每次查询需把b条线段查一遍，m次查询则需计算mb次，复杂度是O(mb)。计算耗时（10^9）
- 思路二：b条线段是固定的，使用线段树，则无需每次都把b条线段都查一遍

示意图（1/3）

已知线段[2,5],[4,6],[0,7]，求点2、4、7在线段上出现的次数



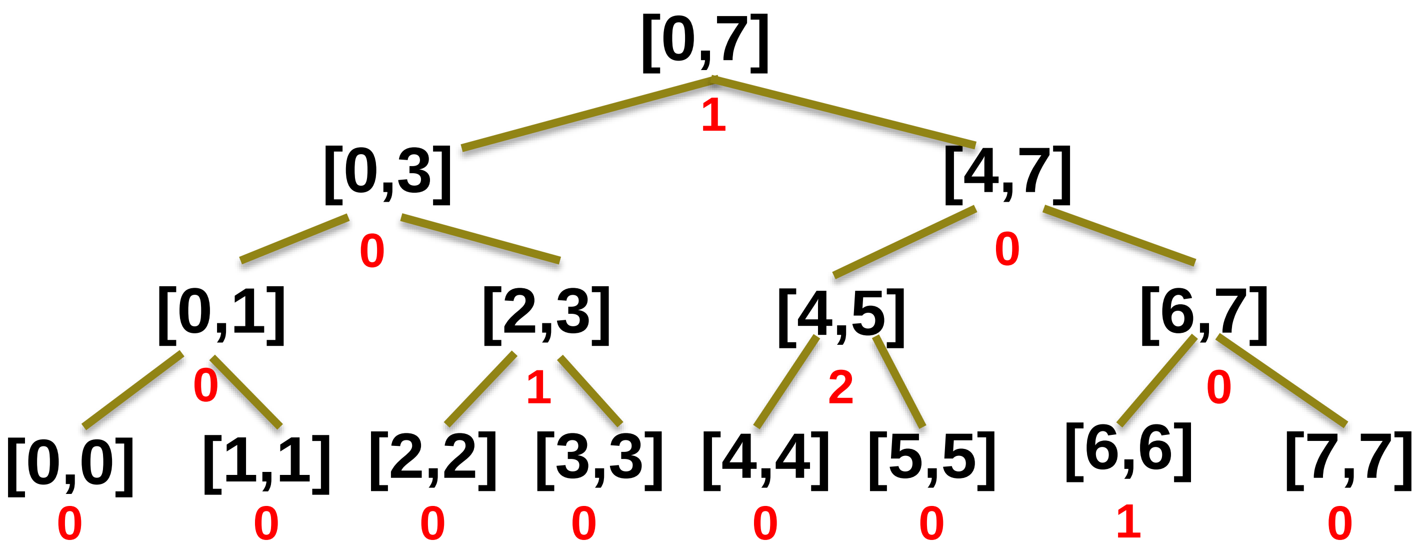
注意：这棵线段树，与前面的略有不同。
实际需根据题意来选择不同的结构



示意图（2/3）

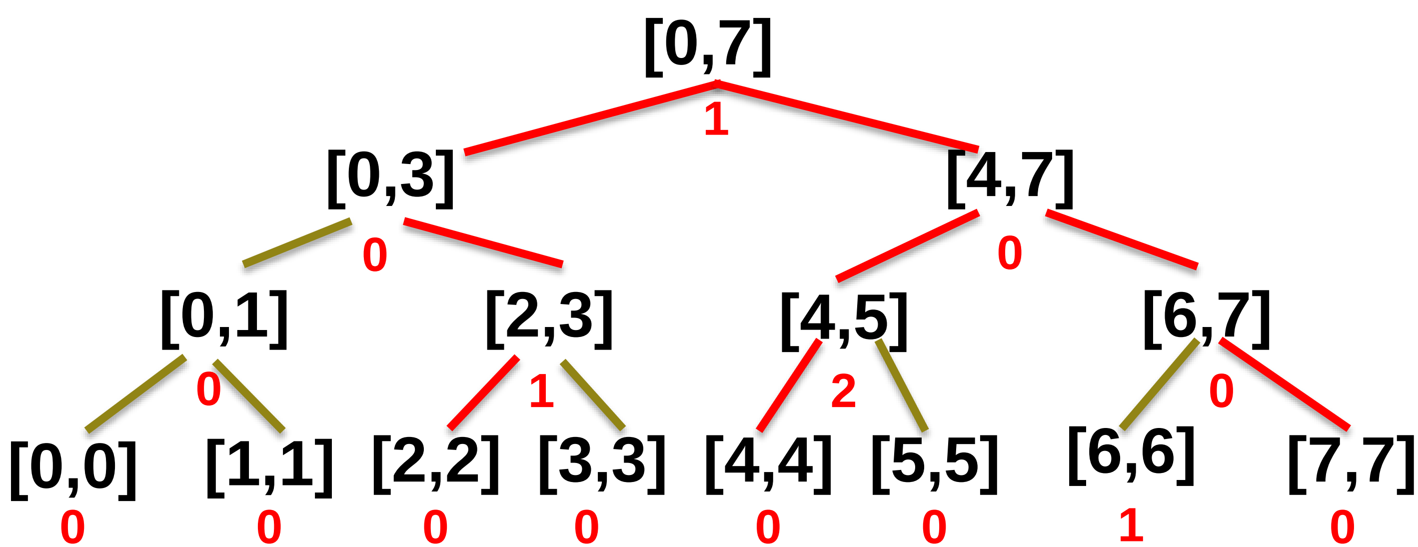
已知线段[2,5],[4,6],[0,7]，求点2、4、7在线段上出现的次数

将三条线段插入到这个线段树中，并保存：有效线段计数



示意图（3/3）

已知线段[2,5],[4,6],[0,7]，求点2、4、7在线段上出现的次数

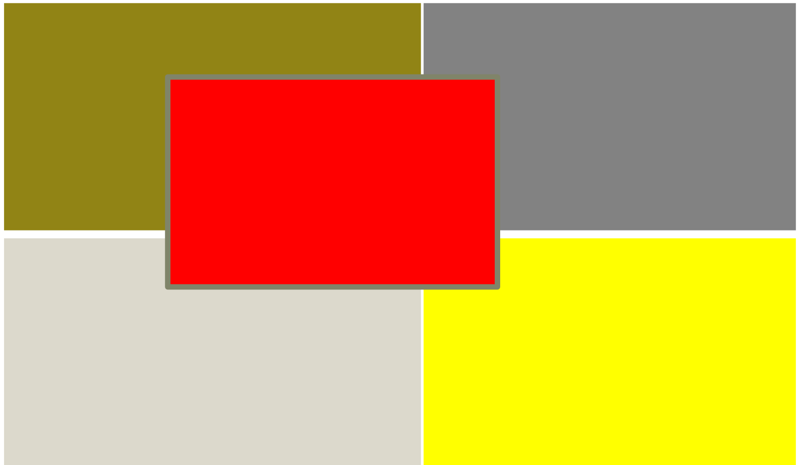


出现次数查询： 2 3 1

插入/查询：每次操作的执行次数为树的深度logN
建树有b次插入，共m次查询，因此时间复杂度为O((b+m)logN)

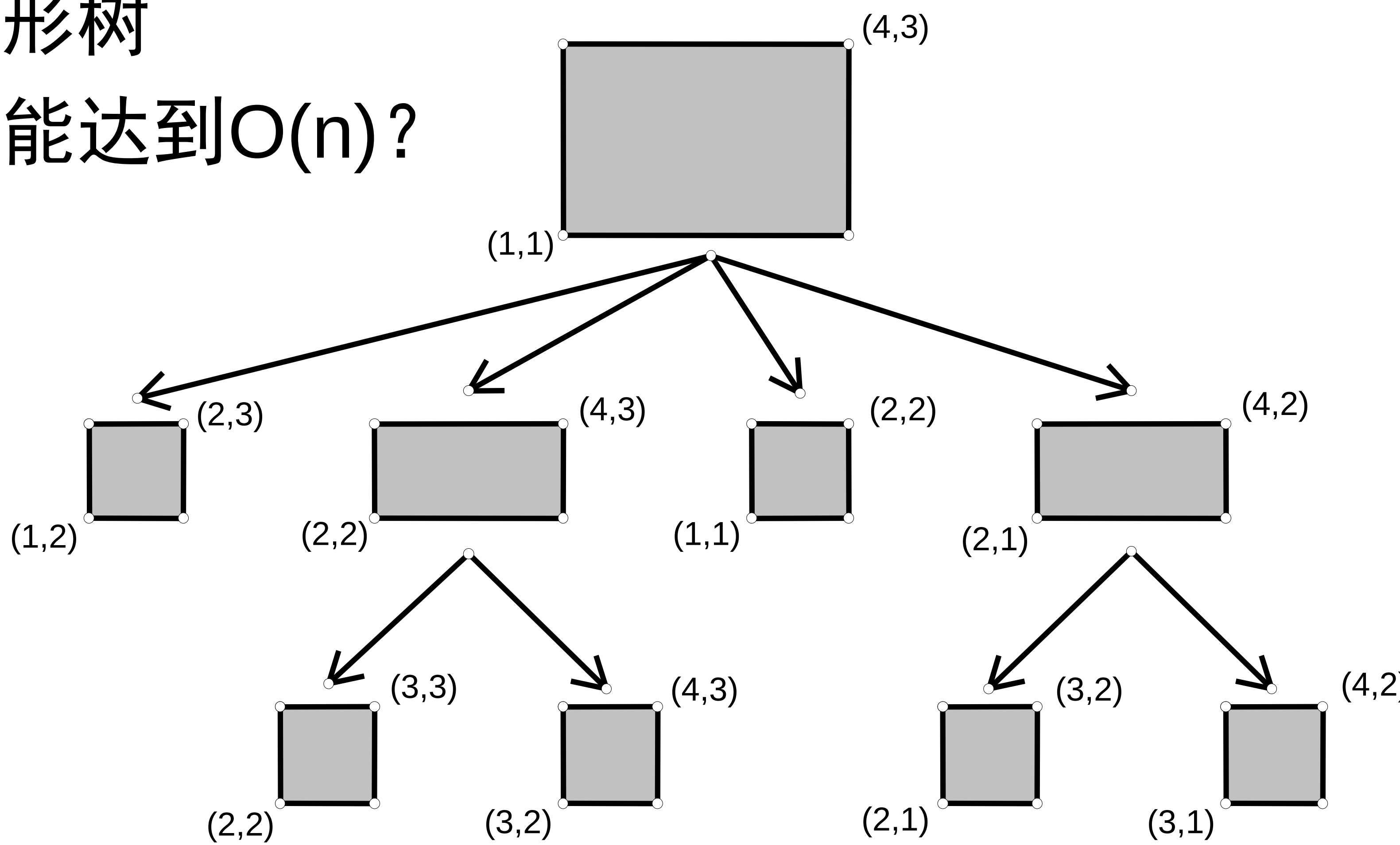
5. 线段树的扩展

- 一个矩阵，可动态修改值和查询最大/最小值、求和
- 把线段树扩展到二维
- 矩形树?

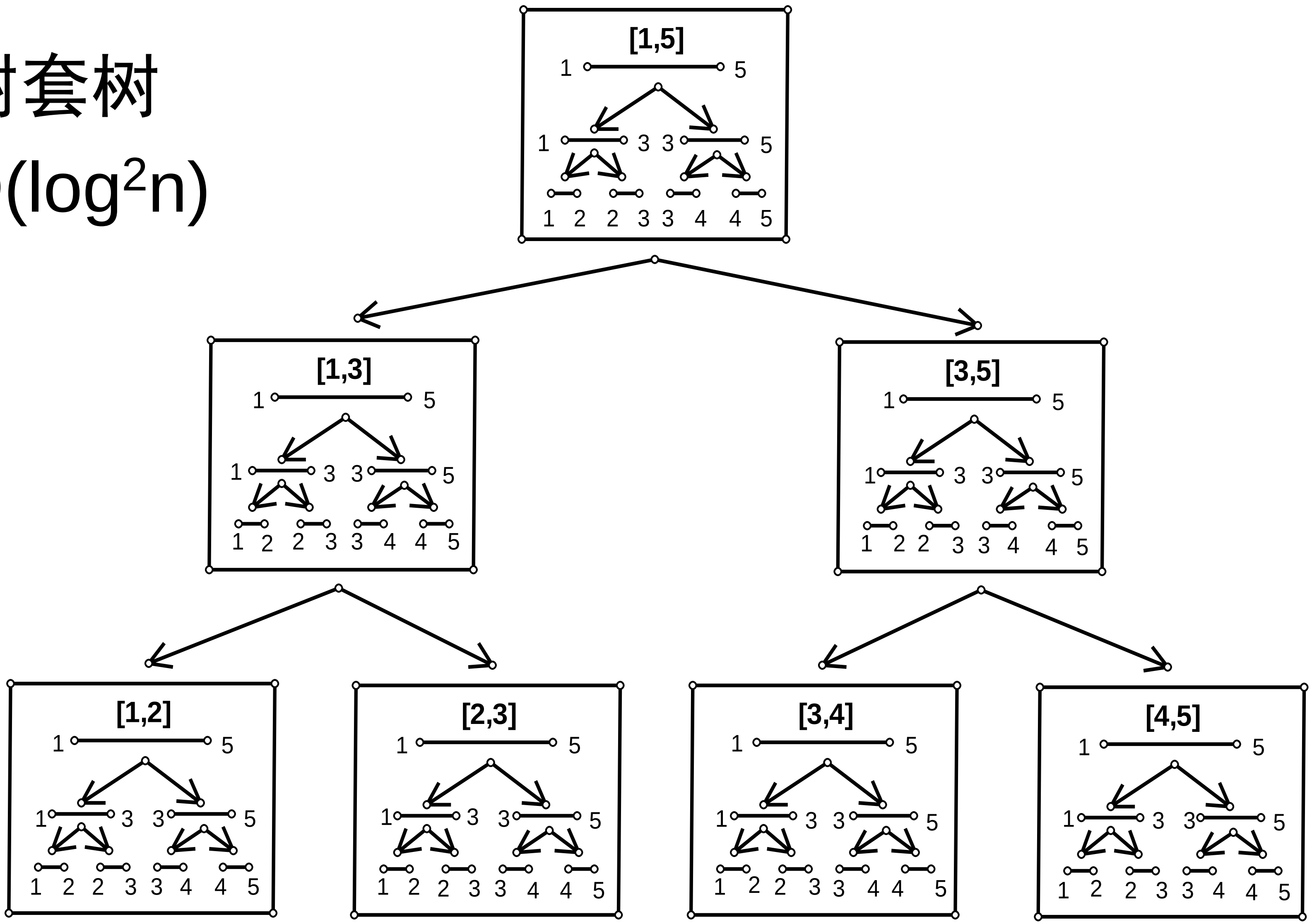


线段树的扩展

- 矩形树
- 可能达到O(n)？



- 树套树
- $O(\log^2 n)$



总结

- 灵活——具体问题具体分析
- 强大——只要你想的到
- 但不是万能的！
- 尽管能够在区间上进行各种操作，但不能插入或者删除区间。
- 父结点的信息要能从子结点完全计算出来。（中位数、众数）

Thanks!