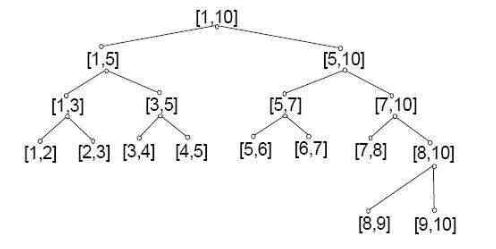
**[初识，线段树和树状数组](http://www.cnblogs.com/DiaoCow/archive/2010/05/07/1729488.html)**

这几天在网上看了很多关于线段树和树状数组的资料，感觉是很重要的数据结构，有必要边学边做下记录。

1.线段树

凡是关于线段树的资料，我想都有这么一张图，的确，它给了人最初的感性认识，其次这幅图在以后分析问题时也很有帮助：



 线段树的基本操作（由于线段树是一棵完全二叉树，且每条线段以类似“二分”的方法处理，因此时间复杂为O(logN)）：

typedef struct  
{

 　　int l , r; //线段左右端点

 　　int cover; //是否被覆盖  
}TreeNode;  
  
TreeNode seg\_tree[N];  
  
/\*创建线段树，叶节点为[a,a+1]\*/  
void CreateSegTree(int p , int a , int b)  
{  
 int m;  
  
 seg\_tree[p].l = a;  
 seg\_tree[p].r = b;  
 seg\_tree[p].cover = 0;  
  
 m = (a + b) / 2;  
 if(m > a)  
 {  
 CreateSegTree(2 \* p , a , m);  
 CreateSegTree(2 \* p + 1 , m , b);  
 }  
}  
  
/\*插入线段\*/  
void InsertSegTree(int p , int a , int b)  
{  
 int m;  
  
 if(seg\_tree[p].l == a && seg\_tree[p].r == b)  
 {  
 seg\_tree[p].cover = 1;  
 return;  
 }  
   
 m = (seg\_tree[p].l + seg\_tree[p].r) / 2;  
 /\*插入左子树\*/  
 if(b <= m)   
 {  
 InsertSegTree(2 \* p , a , b);  
 }  
 /\*插入右子树\*/  
 else if(a >= m)   
 {  
 InsertSegTree(2 \* p + 1 , a , b);  
 }  
 /\*分插两边\*/  
 else   
 {  
 InsertSegTree(2 \* p , a , m);  
 InsertSegTree(2 \* p + 1 , m ,b);  
 }  
}  
  
/\*删除线段(注意当删除一条线段时，要连同它的子树一起删除\*/  
/\*eg:删除线段(1,5)那么像子线段(1,3),(3,5)之类的线段也要删除)\*/  
int DeleteSegTree(int p , int a , int b)  
{  
 int m , nRet;  
  
 if(seg\_tree[p].l + 1 == seg\_tree[p].r)  
 {  
 nRet = seg\_tree[p].cover;  
 seg\_tree[p].cover = 0;  
 return nRet;  
 }  
  
 m = (seg\_tree[p].l + seg\_tree[p].r) / 2;  
 /\*原先完全覆盖\*/  
 if(seg\_tree[p].cover == 1)   
 {  
 seg\_tree[p].cover = 0;  
 seg\_tree[2\*p].cover = seg\_tree[2\*p+1].cover = 1;  
 }  
 if(b <= m)  
 {  
 return DeleteSegTree(2 \* p , a , b);  
 }  
 if(a >= m)  
 {  
 return DeleteSegTree(2 \* p + 1 , a , b);  
 }  
 else  
 {  
 return DeleteSegTree(2 \* p , a , m) && DeleteSegTree(2 \* p + 1 , m ,b);  
 }  
}

(这里的叶子结点是[a,a+1],也就是说线段树中的每个节点都是一个区间，叶节点的区间间隔是1，但是还有一种线段树它的划分方法是，[a , m ] 和 [m+1 , b] 它的叶节点是[a,a]，我们会根据不同的问题选择不同的线段树)