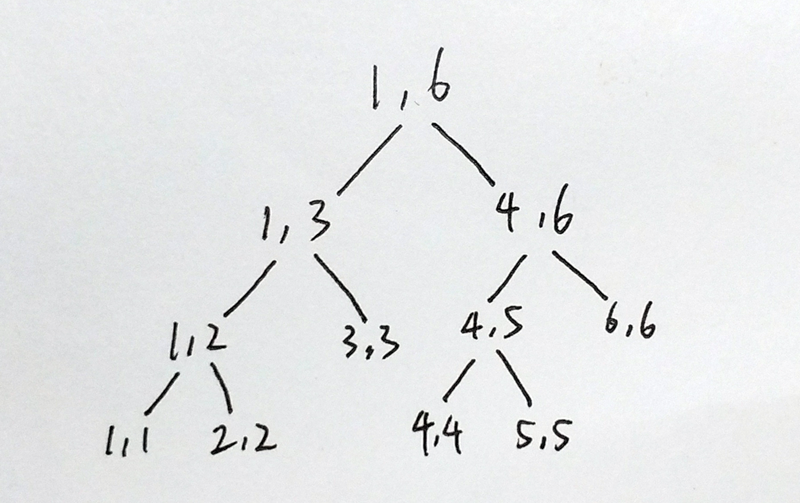
## 概要

线段树，和树状数组一样采用了分段的思想，使得修改与查询都能通过在少量的段上操作，快速地维护结构以及得到结果，线段树。

线段树的每个节点维护了一个区间[L,R]内的信息，若L<R，则将区间分成两半：[L,mid]和[mid+1,R]（mid=(L+R)/2），分别由左子树和右子树维护，若L==R，则本身为叶节点，无需再划分。如果要维护区间[1,n]间的信息，那么根节点区间就为[1,n]。假设区间为[1,6]，相应线段树为：



可见线段树的叶子节点从左向右依次就是单点1-n，并且叶子节点相差不会超过一层，是一颗平衡树，存的时候，依然是根据层次遍历的顺序存在数组里，假设从下标1开始存，那么对于root号节点，左子树编号就是2\*root，右子树就是2\*root+1，可以证明，实际节点数应该开4\*n，就不会越界。

到了现在可能就会有许多疑问了：

这不是一棵完全二叉树，怎么定位到叶子节点呢？答：自顶向下“二分”嘛，当前节点如果不是叶子节点，就看叶子节点在哪个区间范围内，在左区间就找左子树，右区间就找右子树，直到找到相应叶子节点。

是否需要显示地存储每个节点代表的区间呢？其实也没必要的（虽然我写的样例代码中存了，初学为了易懂嘛），线段树的操作基本是自顶向下递归的过程，而根节点代表的区间时知道的，而知道根节点，就知道其子节点的区间范围，所以一路走下去，当前节点代表的区间范围都是知道的，放在函数形参里就行。

接下来主要以区间修改、区间求和为例。

## 建树

建树之初，我们一般只知道区间内每个点的值，即线段树上叶子节点的值，而线段树每个节点维护的是一定区间的值，所以可以递归建树，遇到叶节点时赋值，回溯时，根据左右子节点的值，更新本节点的。

void build(int root,int L,int R)//建一棵没有值的空树

{

x[root].L=L,x[root].R=R,x[root].val=0,x[root].lazy=0;

if(L<R)//还得往下

{

int mid=((L+R)>>1);

build(root<<1,L,mid);

build((root<<1)|1,mid+1,R);

/\*

回溯时，根据左右子节点值更新本节点值

\*/

x[root].val=x[root<<1].val+x[(root<<1)|1].val;

}

else if(L==R)

{

scanf("%lld",&x[root].val);//遇到叶子节点则赋值

}

}

## 区间查询

单点查询本质上就是长为1的区间查询，所以统一在一起。

## 区间修改，lazy标记

单点修改本质上就是长为1的区间修改，所以统一在一起。

和查询一样，仍然需要从根节点，自顶向下，找到查询区间再线段树上对应的所有节点，但现在有了一个问题，假设找到了查询区间的一个子区间[L,R]，整个区间都被修改了，也就是说，其所有子节点，值都会发生变化，那么是不是需要再从该节点出发，修改其所有的子节点呢？如果是，那岂不是比暴力的O(N)修改还要修改更多的点，效率更低么。所以，可以为每个区间，记录一个lazy标记，代表整个区间需要修改的值。