# 线段树总结

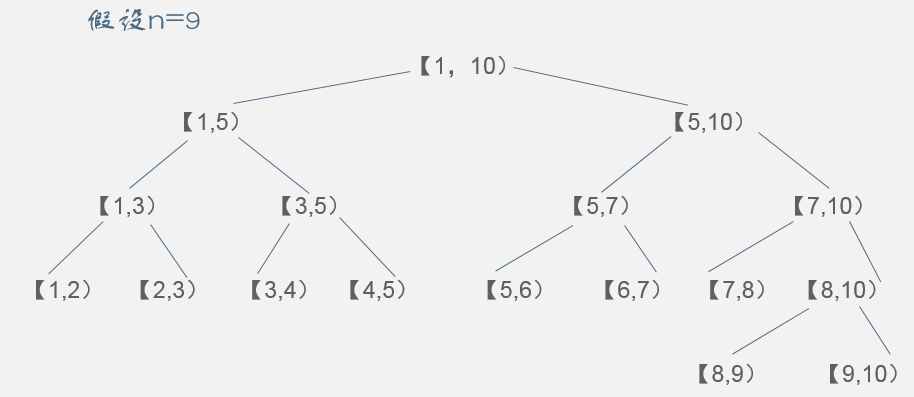
**引入：有一个数组data[1000000]，如果有m个操作(m<=50000)，操作如下：**

1. **修改一个数（加或减）**
2. **求l到r的所有元素的和**

**对于通常的题目，用一个sum[i]数组记录1~i的和，即可将任何的区间和在O(1)的时间之内求出。然而对于操作一，显然就显得很耗时间，至少会花费O(n)，由此可以看出这样的做法是有一定缺陷的，由此线段树便产生了。**

**线段树是一种**[**二叉搜索树**](http://baike.baidu.com/view/389453.htm)**，与**[**区间树**](http://baike.baidu.com/view/3861858.htm)**相似，它将一个区间划分成一些单元区间，每个单元区间对应线段树中的一个叶结点。**

**我们从1~n的区间开始，依次往下二分加入更小的区间，直到区间长度为1**



**如图，这里的左闭右开区间类似于闭区间，表示的元素集合基本相同，我们看到，线段树并不是一个完全二叉树，但是十分类似于完全二叉树，由此可以得到线段树的一般存储方式，用完全二叉树进行存储。**

**操作1中，只能修改一个数，如果要修改线段树的整个区间，要添加n次数，大大增加了线段树的时间。**

**解决方法：定义一个lazy数组，当我们在遍历过程中，发现需要存储的区间已经包含了当前遍历的区间，我们就可以将需要修改的数值通过lazy暂存在此处，直到下一次我们还需要添加数或者在查找区间和时，再调用lazy，将更深处的sum值更新**

**这样就得到了添加元素的通用做法，将所有修改点的操作看作修改长度为1的区间，这样大大增加了算法的普适性**

**更紧凑的存储方式**

**刚刚提到线段树不完全属于完全二叉树，所以用堆的方式存储会引起内存的浪费，以下方式可以做到对内存的一定节省。**

**对于节点区间为[l,r)的结点，我们将其地址映射到(l+r-1)|( (r-1)!=1)，其中(r-l)!=1用来判断r与l差值是否为1。可将使用的空间精确无误地映射到[0,2n-2]中**

**关于开数组的tips：**

1. **对于堆的开法，数组大小至少4n**
2. **对于紧凑的储存方式，可以适当开2n**