# 线段树：

线段树是算法竞赛很常用很重要的数据结构，很多场景需要灵活应用，对它应该有比较深入的研究。线段树常用在动态区间求多次询问下，的区间最大和最小值，区间加和等

## 建树和基本操作的代码实现

具体略思路，代码有两种，链式和数组

链式实现：维护左右指针son[0]son[1]， 类似二叉树，由于线段树是二叉树，可以用数组存储方式，指针的方式浪费了存储地址的内存，推荐下一种。

数组实现：定义结构体数组，1位置代表根节点，对于任意节点编号i i\*2和i\*2+1代表i点的左右，i/2代表父节点，这样很方便的上下查找。而且这样恰好紧密的占满数组的位置。而通过数学计算，可以知道数组开多大，对于维护序列范围

[0-n) ,线段树的结构体数组开到4\*n 即可满足。

## 维护区间和与最值：

最常见的线段树应用

* 建树：走到叶子赋值，递归结束时候更新区间和最值,now->sum=left->sum+right->sum. ,now->min=min(left->min,right->min).
* 区间查询:看查找区间与访问的区间是不是吻合，吻合就直接读取当前访问节点所表示的区间的数据，否则分三类：

1查找的区间在当前区间左边，向左走

2查找的区间在当前区间右边，向右走

3查找的区间跨越了当前区间左右，设mid是当前中点，把查询分裂两部分[l,mid)和[mid,r) 同时向左右查询，结果再合并

* 惰性区间更新 ：某个区间所有值+val ,如果每次都去更新叶子节点的所有值，效率太低，一次就是nlogn,应该惰性标记，在结构体单开属性lz，表示待更新的值 ，用查询的思路去标记lz,在以后的操作(包括更新和查找)，访问到lz标记不是0的节点应该下放(下方应该是单门的一个函数)
* 暴力更新：复杂度较高，但无法用惰性维护的属性可以考虑它们本身变化频率，如果变化频率较低，可以暴力更新，适用于区间整数根号和等。

## 权值线段树：

类似建立索引,计数排序的思路,对于给定序列A,线段树叶子节点[l,r)不是记录A[l]本身。应该是：对于叶子结点tree[now].r==tree[now].l+1 ,tree[now].val代表了A[l]这个数的个数(也可能是其他属性)。这样线段树区间和[1,n)的意义就是:A序列小于n的数个数..在求逆序对有重要应用。求第k大元素 前驱 后继也可以用权值线段树，但是需要离散化和离线处理，不如平衡搜索树好。

## 区间染色:

类似求最值的区间修改，但染色在修改时,不是增加某个值,而是变成每个值。惰性标记lz代表区间要修改成什么，因为时间靠后的修改会覆盖时间靠前的修改，每次更新lz直接赋值即可。对于真正修改区间时，如果这个区间的子区间颜色不同，就更新成-1，表示这个区间颜色不确定，否则就要更新成相应颜色。

## 连续段长:

模型是：给定长度是n的序列A[]，去掉序列某些值使序列不连续，问你某个A[i]所在连续区间的左右端点或区间长度。

这里如果想维护每A[i]所在连续段的左右端点是可以的，这是我最初的想法，这样做比较麻烦，因为每次删除或者加入一个A[i]要用2个懒惰标记更新其他区间的值，时间常数大代码也麻烦。在网上看到正招是：对于线段树节点now,维护noe内部最大连续长度len,从now.l开始的连续长度x，从now.r开始的连续长度y.这样只有每区间合并的操作，不涉及区间更新了。

和并思路是：父节点的len是左孩子的len 右孩子的len  左孩子的右端长度+右孩子的左端长度 这三个的最大值

双亲的x  如果左孩子整个区间都是连续的话  x就等于左孩子的len加上右孩子的左端长度了

双亲的y  如果右孩子整个区间都是连续的话  y就等于右孩子的len加上左孩子的右端长度了

查询某个编号i的元素所在连续段长度：从根节点开始找：根节点连续，根长度就是答案；如果i在左孩子的y或者右孩子的x内部，那么x+y就是答案了；否则在那个子节点就往哪个节点走继续找。

修改区间长度：用懒惰标记即可。

## 面积并(hdu1255或poj1177)：

问题：给定n个矩形的,顶点位置是已知的浮点数，且每条线段和坐标轴平行，求被矩形覆盖了1次以上的面积总和，以及被覆盖2次以上的面积总和。

总体思路是把边当做线段，用扫描线横或竖着扫，遇到一个线段就计算已经得到的面积并。详细来说以竖着从下往上为例：

离散化横着线段的x坐标，这里可以用map<double,int>建立的索引，比较方便。

ind[x]代表浮点数x离散化之后的整数数值，todou [i]表示离散化后整数数值是i所对应的浮点数。线段树维护s1代表被覆盖了1次以上的宽度总和，s2被覆盖了2次以上的宽度总和，col是被覆盖的次数，初始值它们都是0。这棵树与以往线段树不同，没有懒惰标记因为它只查询根节点数值，而区间修改直接通过两个儿子得到。设sdis()是当前区间最大值，对于叶子结点，覆盖次数是0：s1=s2=0 ;

1：s1=sdis() s2=0; 2以上: s1=s2=sdis()

对于非叶子结点，覆盖次数是0: s1=左.s1+右.s1 s2=左.s2+右.s2;

1：s1=sdis() s2=左.s1+右.s1; 2以上：s1=s2=sdis();

找覆盖两次以上的面积方法：

关于覆盖次数col的区间更新，就是线段树区间结点执行上述操作。

扫描线找到一个线段,若它是某个矩形下方的线段，d=上个线段到当前线段的距离,就把结果加上d\*root.s2,然后区间更新把线段映射到离散数值[l,r),区间加一

若它是某个矩形上方的线段，d=上个线段到当前线段的距离,就把结果加上d\*root.s2,然后区间更新把线段映射到离散数值[l,r),区间减一

## 题目和巧妙的用法：

### 51nod1019(逆序对):

权值线段树的典型应用

逆序问题需要先离散化，就是排序从新赋值使得数据范围缩小，

完了用建立索引的方法，只不过是在线段树上建立索引。

设序列是A[]

C[x]代表A数组里x=A[i]的数的个数，

那么求和函数getsum[x]就代表小于等于x=A[i]的数的个数。

此时如果用暴力的方法做，只要求出A数组每个元素前面，数值比它大的数个数就可以。如果A序列是一个个加入在末尾的，每加入一个数就找比它大的数个数就可以了。如果在logn时间找出它大的数个数？

那么每次插入可以表示为：insert(A[i],1)也就是A[i]这个数的个数+1即可

在插入A[i]同时，每插入一个数A[i]就求一次i- getsum[A[i]],得到了当前比A[i]大的数的个数，由于A[i]是新插入的数，总是在末尾，就求得了当前情况下的逆序对数，把所有的逆序对数加和就是总数。复杂度n\*log(n)

### Hdu6315:

一道复杂更新的题目，<http://acm.hdu.edu.cn/showproblem.php?pid=6315>，

难点在于怎么查找区间和，因为是取整，无法向普通的线段树一样维护区间和。

但是注意到，某些情况区间全都+1 是不影响区间和的，比如在B数组1 5 2 4 3的2 4 3几个数，A数组都是,把A数组+1并不能使区间内任意A[i]/B[i]增加，而且能注意到，B序列是[1,n]的排列的某一个,意味着哪怕每次更新[1,n]大区间，也只是较小的数1,2这样的频繁更新，最坏总更新次数是N\*调和级数前n项和，调和级数是：1/1+1/2+1/3+…+1/n,而调和级数前n项和约等于logn.意味着A[i]/B[i]增加次数不会太多也是n\*logn级别的

正招是用线段树维护sum,minb,maxa,lz , sum不用说代表A[i]/B[i]区间和；minb代表B数组区间最小值；maxa代表a数组区间最大;lz是mina的懒惰标记，这个招法前面讲了。初始建树时候除了minb其余属性都是0

每次区间a数组+1,就用懒惰标记,标标记的时候，比较minb和maxa，如果这个区间maxa>=minb了，就说明需要A[i]/B[i]该增长了,没有什么好办法，就是暴力往下访问子树，更新叶子结点的minb，让它加上B[i],以及更新sum值，这里加一即可，不放心先用A[i]/B[i]得一个数，加到sum上。

往下更新需要特别注意，不要真的用二叉树遍历方法遍历所有节点，而是时刻判断maxa>=minb的条件不行就不走了。不能自作聪明加什么优化，比如每次走最小的minb，这样不行，因为有懒惰标记存在可能一次增长多个minb.

这个过程时刻不要忘了懒惰标记的向下延伸，和函数结束更新sum,minb,maxa

### P3373：

求区间和，但是既有区间加法更新，又有区间乘法更新。

凡是区间更新的情况就要往懒惰标记上想，显然应该设置2个懒惰标记。那这道题只要用常规的惰性更新不就解决了吗，其实不然，加法和乘法操作是互相影响的，你先加再乘结果显然不等于先乘再加。这里有个很好的思路就是自定义一个适当顺序，这里是先更新乘法再更新加法，但是为了确保结果正确，更新乘法时候要顺便把加法的懒惰标记增加，代表了这次下方乘法标记而对加法标记的影响，这就保证了结果的正确，本题中：x先放乘法标记后方加法，加法标记照常下放，而乘法除了更新子节点的值，还要对子节点加法标记更新为:自己乘上当前节点的乘法标记值