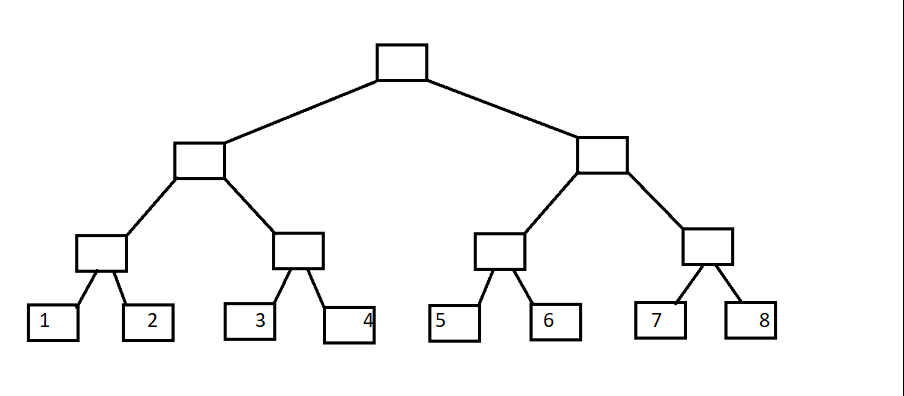
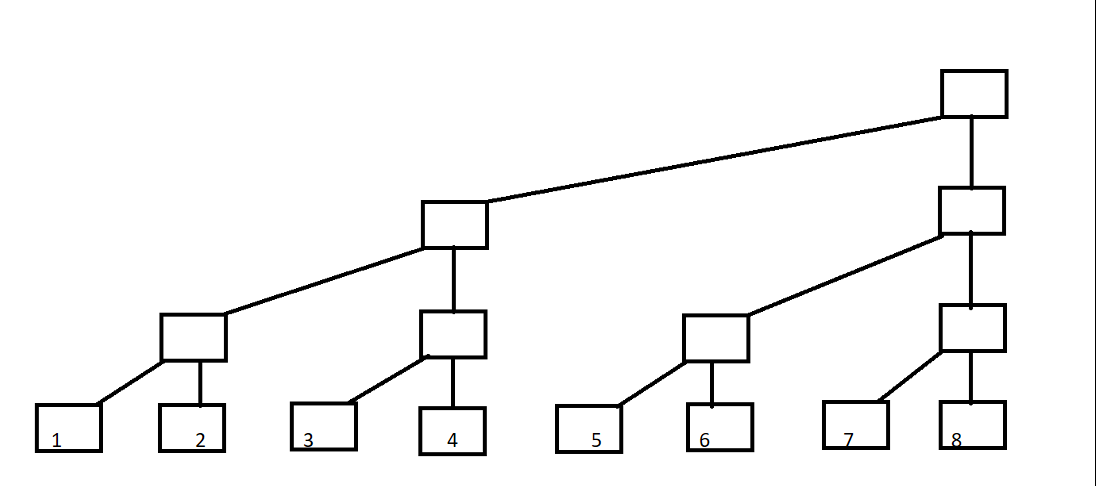
# 树状数组

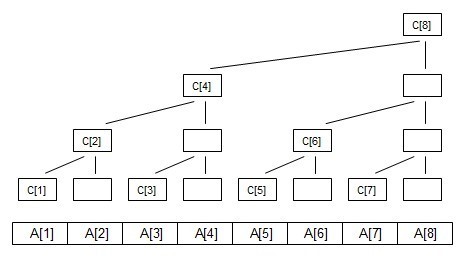
先说二叉树，对于数组A这样分配，分给叶子结点



变形



定义和A有关的C数组，它的大小和A一样但是。。。。，存的值和A不一样



C[i]代表 子树的叶子结点的权值之和

C[1]=A[1];

C[2]=A[1]+A[2];

C[3]=A[3];

C[4]=A[1]+A[2]+A[3]+A[4];

C[5]=A[5];

C[6]=A[5]+A[6];

C[7]=A[7];

C[8]=A[1]+A[2]+A[3]+A[4]+A[5]+A[6]+A[7]+A[8];

将C[]数组的结点序号转化为**二进制**

1=(001)      C[1]=A[1];

2=(010)      C[2]=A[1]+A[2]=C(1)+A(2);

3=(011)      C[3]=A[3];

4=(100)      C[4]=A[1]+A[2]+A[3]+A[4]=C(2)+C(3);

5=(101)      C[5]=A[5];

6=(110)      C[6]=A[5]+A[6];

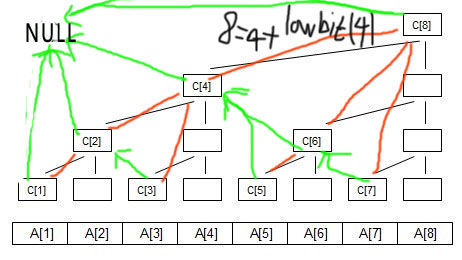
7=(111)      C[7]=A[7];

8=(1000)    C[8]=A[1]+A[2]+A[3]+A[4]+A[5]+A[6]+A[7]+A[8]=C(7)+A(8);

**C[i]=A[i-2^k+1]+A[i-2^k+2]+......A[i]; （k为i的二进制中从最低位到高位连续零的长度）例如i=8时，k=3;**

引入lowbit(x)

lowbit(x) 其实就是取出x的最低位1 ， 换言之**lowbit(x)=2^k**



这个数据结构从上向下看，只储存了每层的右节点。

并且通过上图红线把C数组相连后，形成树形关系，红线代表了当前节点向上指向哪里(x+lowbit(x))，绿色节点代表向下指向哪里(x-lowbit(x))，从下到上节点可以通过加上lowbit(x)的方法得到，我们把这种用红线绿线连接的新的树形数据结构，叫树状数组

父节点(编号为x)的值=所有自己节点值+A[x]的值

## 插入更新操作：

那么插入和更新同一个操作

在x的下标变成一个数y,把原本C[x]记作y0;那么就是y0+ y-y0;也就是

C[x]+=y-C[x];然后更新父节点loubit(x)直到更新到根；当初始化C数组是0时，插入可以简写成C[x]+=y

## 查找前缀和：

本质就是沿着当前节点向下找，x=x-lowbit(x);

每个找到相加，由于树状数组的特殊性，和树不一样，这个向下找并不是找到他的最近儿子，事实上树状数组是有向图，x+=lowbit(x)是找父节点，但是x-=lowbit(x)并不能返回刚才那个点。比如比如4+lowbit(4)=8;但是8-lowbit(8)确是1

## 性能分析：

查询时间复杂度O(logn)和线段树一样，建树操作n也和线段树一样，但时间常数小，比线段树快几倍的速度，运行占用空间小于线段树，代码量也比较小，好写容易记，时间常数小。

## 不足：

它和线段树解决同样的问题，但是树状数组能坐的线段树也能做，线段树能做的树状数组不一定能做，可扩展性不如线段树强大，适合做那种写烂了的模板题写起来代码量小省时间，原理和线段树不同，但功能上可以认为是低配线段树。

# 树状数组解决问题的模型：

## 带有单点修改的区间和问题：

举的例子就是求和问题，方法是维护前缀和，求区间[a,b]和是求

getsum(b)-getsum(a-1),插入和更新是一个方法，即：某个节点的值加x,更新父亲节点

## 带有区间修改的单点询问问题：

给你一个数组A，有2个操作：1区间[l,r]加上某个数，2询问某个数的值。

树状数组是不支持区间修改的，所以要想办法模拟，方法用树状数组维护A的差分数组，设数组A={1,6,8,5,10}，那么差分数组a={1,5,2,-3,5},通过数学计算很容易知道，区间[l,r]修改A，等价于单点修改a数组的a[l]和a[r+1]；对于询问某个A的值，等价于求a数组前缀和

## 求逆序问题：

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，只不过是在树状数组上建立索引。

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)