树状数组与线段树基础

——19计科朱俊杰

内容部分参考《算法竞赛进阶指南》

1.前置知识回顾

算术位运算

> 简单回顾一下算术位运算
>
> > **与** and，& 1&1=1,1&0=0,0&1=0,0&0=0.
> >
> > **或** or，| 1|1=1,1|0=1,0|1=1,0|0=0.
> >
> > **非** not，~ ~1=-2, ~0=-1.
> >
> > **异或** xor，^ 1^1=0, 1^0=1, 0^1=1, 0^0=0.

如何求一个正整数的最低位?

即把一个正整数x转化为二进制数，找到最靠后的那个1所对应的大小。

比如$(6)_{10}$=$(110)_2$,它的最低位就是$(10)_2$=$(4)_{10}$。

首先最朴素的想法就是挨个尝试每一位，即枚举x与2^n（n从1开始枚举）进行按位与操作得到的第一个结果不为0的2^n，即为最低位。

这样做的效率不高，给出一个效率较高的算法。

最低位为**x&(-x)**

lowbit(x)=x&(-x)

证明，位运算是通过补码来实现的，负数的反码等于其正数的原码的反码再加一，即$(6)_{10}$=$(110)_2$,

$(-6)_{10}$=$(001)_2$+$(001)_2$=$(010)_2$

$(010)_2$^$(110)_2$=$(010)_2$.

可以发现变为反码后 x 与反码数字位每一位都不同， 所以当反码加1后神奇的事情发生了，反码会逢1一直进位直到遇到0，且这个0变成了1，所以这个数最后面构造了一个 100... 串。 由于是反码，进位之后由于1的作用使进位的部分全部取反即与原码相同，所以可以发现 lowbit 以前的部分 x 与其补码即 -x 相反， lowbit x 与 -x 都是1，lowbit 以后 x 与 -x 都是0 所以 x&-x 后除了 lowbit 位是1，其余位都是0。符合条件。

移位运算

> **左移**
>
> > 在二进制表示下把数字同时向左移动，低位以0补充，高位越界后舍去。
> >
> > 1<<n=2^n,n<<1=$2n$.
>
> **右移**
>
> > 在二进制补码标下把数字同时向右移动，高位以符号位填充，低位越界后舍去。
> >
> > n>>1=$\lfloor \frac{n}{2.0} \rfloor$
> >
> > **注**：当n<0,n>>1!=n/2.（-3>>1)=-2,-3/2=-1;

对于一个正整数n,**n<<1|1=2*n+1**,**n<<1=2*n**

离散化

> 这是排序算法的一个应用，通俗地，“离散化”就是将一个无穷大集合中的若干个元素映射到有限集合便于统计的方法。
>
> 具体地说，假设问题涉及到int范围内的n个整数a[1]到a[n]，这n个整数可能有重复，去重以后共有m个整数。我们把每个整数a[i]用一个1~m之间的整数表示，并保持它们的大小关系不变。

```
void discrete(){//离散化
    sort(a+1,a+1+n);//对原先的数组进行排序以确保大小关系不变
    m=0;
    for(int i=1;i<=n;i++)
        if(i==1||a[i]!=a[i-1])
            b[++m]=a[i];
}
int search(int x){//查询数组a中的x映射为1~m之间的哪个整数
    return lower_bound(b+1,b+1+m,x)-b;
}
```

2.树状数组

树状数组的思想

树状数组是基于二进制思想的，我们可以将一个正整数x分解为若干个不重复的2的幂次和，即

x=$2^{i_1}+2^{i_2}+2^{i_3}+\cdots 2^{i_m}$

假设i_1>i_2>…>i_m，则区间[1,x]我们可以最多分成$log_2(x+1)$个小区间：

1.长度为2^{i_1}的小区间[$1,2^{i_1}$]

2.长度为2^{i_2}的小区间[2^{i_1}+1,2^{i_1}+2^{i_2}]

……

m.长度为2^{i_m}的小区间[2^{i_1}+$2^{i_2}+\cdots+2^{i_{m-1}}$+1,$2^{i_1}$+$2^{i_2}+\cdots+2^{i_m}$]

这些区间的共同特点是：若区间结尾为R，则区间长度等于R的”二进制分解“下最小的二的幂次，即lowbit（R）。例如 x=7=2^2+2^1+2^0,区间[1,7]可以被分解为三个小区间长度分别是lowbit（4）=4，lowbit（6）=2，lowbit（7）=1。

树状数组的基本用途

它的基本作用是**维护序列的前缀和**,对于给定的序列a，我们建立一个数组c，其中c[x]**保存序列a的区间[x-lowbit(x)+1,x]中的所有数的和**，即$\sum_{i=x-lowbit(x)+1}^{x} a[i]$.

事实上，数组c可以看做一个如下图所示的树形结构，图中最下边一行是N个叶子结点（N=16），代表数值a[1~N]。该结构满足以下性质：

1.每个内部节点c[x]保存以它为根的子树中所有节点的和。

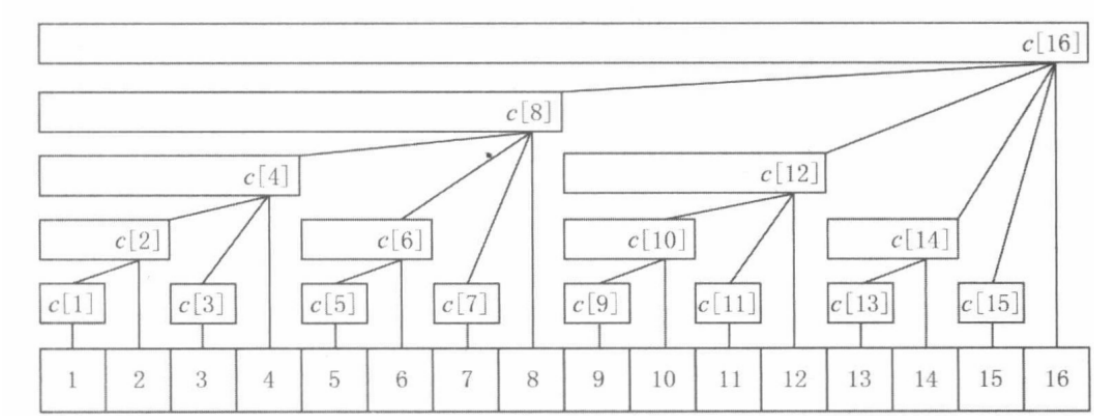
2.每个内部节点c[x]的子节点个数等于lowbit(x)的位数。

3.除树根外，每个内部节点c[x]d的父亲节点是c[x+lowbit(x)]。

4.数的深度为O(logN)。

如果N不是2的整次幂，那么树状数组就是一个具有同样性质的森林结构。

树状数组的结构图



树状数组的基本操作有两种，一是查询前缀和，二是进行单点修改。

树状数组的查询前缀和

前缀和，即求序列a第1~x个数的和。按照之前的思路，需要求出x的二进制表示中每个等于1的位，把[1,x]分成小区间，而小区间的信息已经保存在数组c中，例如我们需要求解序列[1,7]的和，sum[1,7]=sum[7,7]+sum[5,6]+sum[1,4]=c[7]+c[6]+c[4];代码实现如下：

```
int ask(int x){
    int ans=0;
    while(x){
        ans+=c[x];
        x-=lowbit(x);
    }
    return ans;
}
```

这段代码的时间复杂度为O（logN）。

如果要求a序列[l,r]的区间和，只需要计算ask（r）-ask（l-1）即可。

树状数组的单点修改

单点修改，顾名思义我们需要对序列a中的某个数a[x]进行修改，将其值增加y，对于维护的前缀和数组c我们需要进行相应的修改以确保区间和的正确性。例如我们对于大小为15的树状数组进行对a[6]+1的操作，c数组对应需要改变，c[6],c[8].代码实现如下：

```
void insert(int x,int y){
    while(x<=N){
        c[x]+=y;
        x+=lowbit(x);
    }
}
```

不难看出这部分代码的时间复杂度也是O（logN）的，在对树状数组初始化时需要进行N次操作。

对于树状数组的整体的时间复杂度，假设数组的大小为N，进行修改与查询操作的总和为M则代码的时间复杂度为O（（N+M）logN）。

类似地，基于差分思想，树状数组还可以维护区间修改和单点查询。

树状数组求逆序对数量

树状数组计算逆序对的数量，之前我们在学习归并排序的时候，归并排序能够在O(NlogN)内求出逆序对的个数，树状数组也能够实现。

将我们需要计算逆序对的数组（如果数据范围过大需要先离散化）倒序维护一个空的树状数组，在每个数更新之前累计求在它之前数的数量。例如序列a={2,3,5,4,1}。步骤如下

ans+=ask（1），insert（1,1）；ans=0

ans+=ask（4），insert（4,1）；ans=1

ans+=ask（5），insert（5,1）；ans=3

ans+=ask（3），insert（3,1）；ans=4

ans+=ask（2），insert（2,1）；ans=5

所以逆序对的数量为5

例题逆序对

求逆序对的数量

多组输入，每个测试用例第一行为一个整数n（n<500000）表示数组的长度
下面n行每一行包含一个整数0≤a[i]≤999,999,999，即数组中第i个元素的值。

当n=0时结束输入

如何使用树状数组来解决它呢?

原题链接

3.线段树基础

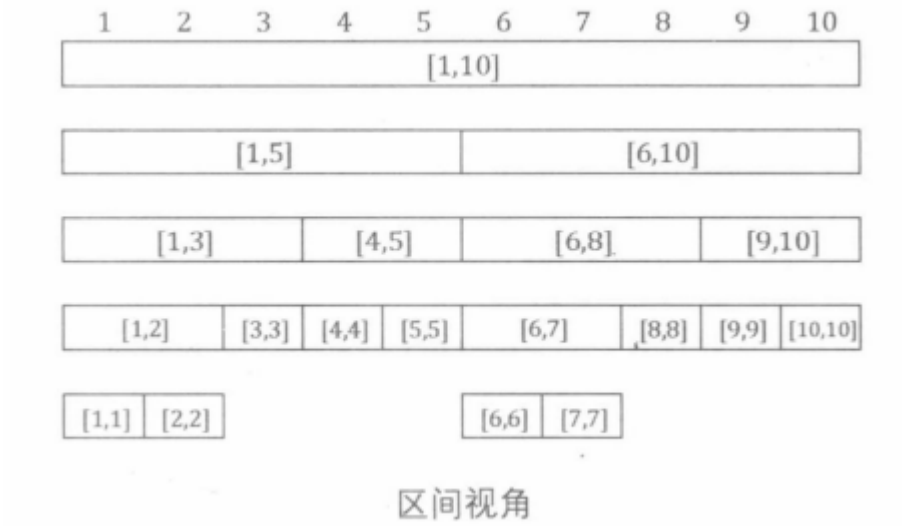
线段树的思想

线段树（segment tree）是一种基于分治思想的二叉树结构，用于在区间上进行信息统计。与按照二进制思想的树状数组相比，线段树是一种更加通用的结构：

> 1.线段树的每一个节点都代表一个区间。
>
> 2.线段树具有唯一的根节点，代表的区间是整个统计范围，如[1,N]。
>
> 3.线段树的每个叶子节点都代表一个长度为1的元区间[x,x]。
>
> 4.对于每个内部节点[l,r],它的左节点是[l,mid]，右节点是[mid+1,r],mid=(l+r)/2。

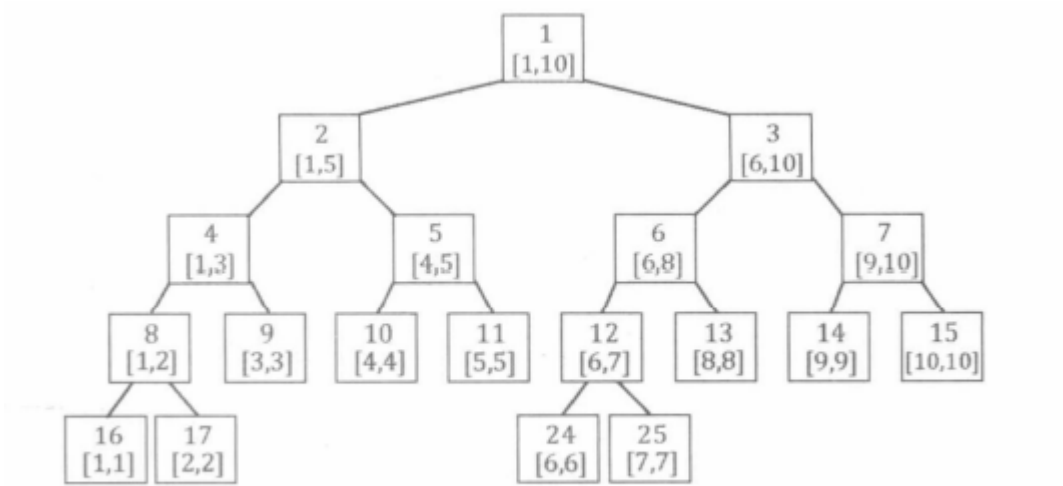
线段树的举例

以区间[1,10]为例，它的线段树的结构如下：



线段树的节点标号

一般以根节点为1号节点，对于其他节点，假设根节点标号为rt，左子节点的标号为rt<<1，右子节点的标号为rt<<1|1。



二叉树视角

线段树的构建及空间规划

一般线段树我们开4倍的空间，这是足够用的。

以下为**维护区间最大值**的线段树构建代码。

```
#define ls rt<<1
#define rs rt<<1|1
const int N=1e5+5;//区间最大为1e5
int a[N],t[N<<2];//t数组维护区间信息
void push(int rt){
    t[rt]=max(t[ls],t[rs]);更新节点信息
}
void build(int rt,int l,int r){
    if(l==r){
        t[rt]=a[l];
        return;
    }
    int mid=l+r>>1;
    build(ls,l,mid);//递归建树
    build(rs,mid+1,r);
    push(rt);//将信息由子节点更新到父节点
}
build(1,1,n);//调用，将a数组中的1~n元素建立线段树
```

线段树的单点修改

线段树还支持单点修改，还是以维护区间最大值的线段树为例。

单点修改的操作如“C x v”的指令，表示将a[x]的值修改为v。我们需要找到线段树中元节点[x,x]修改并将其信息更新到包含它的所有区间中。我们可以递归的找寻这个节点，然后回溯的过程中对信息进行更新。

代码如下：

```
void insert(int rt,int l,int r,int x,int v){
    if(l==r){
        t[rt]=v;
        return ;
    }
    int mid=l+r>>1;
    if(x<=mid){//修改的元节点包含在左节点
        insert(ls,l,mid,x,v);
    }
    else insert(rs,mid+1,r,x,v);
    push(rt);//更新信息
}
```

线段树的区间查询

区间查询是形如“Q l r”的指令，表示需要查询区间[l,r]的最大值。我们只需要从根节点出发，递归执行一下过程：

> 1.若[l,r]完全覆盖了节点代表的区间，则立即回溯，将该节点的信息作为候选答案。
>
> 2.若左子节点与[l,r]有重叠部分，则递归访问左子节点。
>
> 3.若右子节点与[l,r]有重叠部分，则递归访问右子节点。

代码如下：

```
int query(int rt,int l,int r,int L,int R){//[L,R]表示查询区间
    if(L<=l&&r<=R){//1
        return t[rt];
    }
    int mid=l+r>>1;
    int ans=-0x3f3f3f3f;//负无穷
    if(L<=mid) ans=max(ans,query(ls,l,mid,L,R));//2
    if(R>=mid+1) ans=max(ans,query(rs,mid+1,R));//3
    return ans;
}
```

线段树的用途

线段树的时间复杂度是**O(logn)**级别的，对于解决区间问题有着显著的效果，能用线段树解决问题的关键在于区间信息的合并，否则是无法使用线段树解决的，像区间的最值，区间和等都可以用线段树来维护，甚至区间的gcd也是可以使用线段树来维护的，线段树对于区间众数一类不利于区间合并信息的信息是没有办法维护的。

线段树例题养花

wr养了很多花，每盆花都有一个美观值，照顾的好它的美观值会上升，照料不好会下降，他想知道某段连续的花的美观值之和是多少？但是他算术不好，需要你帮帮他。

第一行一个整数T，表示有T组测试数据。
每组测试数据的第一行为一个正整数N (N<=50000)，表示Lily有N盆花。
接下来有N个正整数，第i个正整数ai (1<=ai<=50) 表示第i盆花的初始美观值。
接下来每行有一条命令，命令有4种形式：
（1）Add i j, i和j为正整数，表示第i盆花被照料的好，美观值增加j (j<=30)
（2）Sub i j, i和j为正整数，表示第i盆花被照料的不好，美观值减少j (j<=30)
（3）Query i j, i和j为正整数，i<=j，表示询问第i盆花到第j盆花的美观值之和
（4）End，表示结束，这条命令在每组数据最后出现
每组数据的命令不超过40000条

原题链接