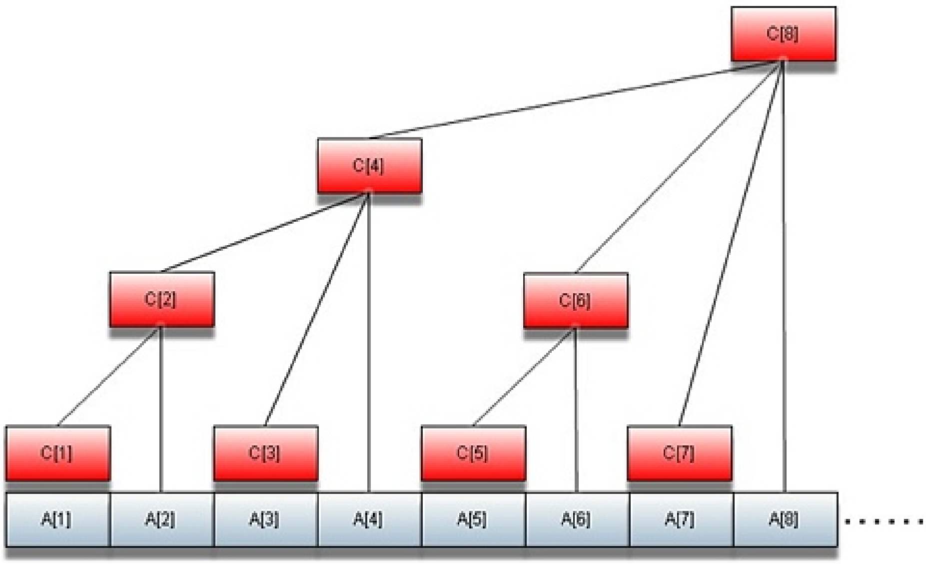
树状数组入门教程

【引言】

在解题过程中，我们有时需要维护一个数组的前缀和 S[i]=A[1]+A[2]+...+A[i]。但是不难发现，如果我们修改了任意一个 A[i],S[i]、S[i+1]...S[n]都会发生变化。可以说，每次修改 A[i]后，调整前缀和 S 在最坏情况下会需要 O(n)的时间。当 n 非常大时，程序会运行得非常缓慢。因此，这里我们引入“树状数组”，它的修改与求和都是 O(logn)的，效率非常高。

【理论】

为了对树状数组有个形象的认识，我们先看下面这张图。



如图所示，红色矩形表示的数组C就是树状数组。这里，C[i]表示A[i-2^k +1]到A[i]的和，而k则是i在二进制时末尾 0 的个数，或者说是i用 2 的幂方和表示时的最小指数。当然，利用位运算，我们可以直接计算出 2^k =i and (i xor (i-1))**或i and (-i)**

同时，我们也不难发现，这个 k 就是该节点在树中的高度，因而这个树的高度不会超过 logn。所以,当我们修改 A[i]的值时，可以从 C[i]往根节点一路上溯，调整这条路上的所有 C 即可，这个操作的复杂度在最坏情况下就是树的高度即 O(logn)。

另外，对于求数列的前 n 项和，只需找到 n 以前的所有最大子树，把其根节点的 C 加起来即可。不难发现，这些子树的数目是 n 在二进制时 1 的个数，或者说是把 n 展开成 2 的幂方和时的项数,因此，求和操作的复杂度也是 O(logn)。

树状数组C，其中C[i]=a[i-2^k+1]+……+a[i](k为i在二进制形式下末尾 0 的个数)。由c数组的定义可以得出：

c[1]=a[1]

c[2]=a[1]+a[2]=c[1]+a[2]

c[3]=a[3]

c[4]=a[1]+a[2]+a[3]+a[4]=c[2]+c[3]+a[4]

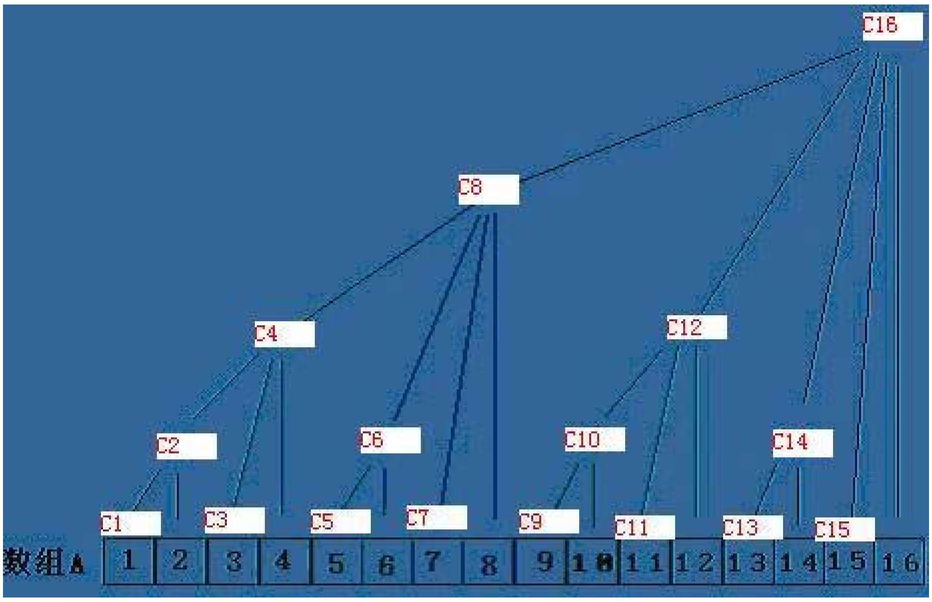
c[5]=a[5]

c[6]=a[5]+a[6]=c[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]=c[4]+c[6]+c[7]+a[8]
………………
c[16]=a[1]+a[2]+………………………+a[16]=c[8]+c[12]+c[14]+c[15]+a[16]

c 数组的结构对应一棵树，因此称之为树状数组。

对应如下图形：



在最后，我们将给出一些树状数组的实现代码，希望读者能够仔细体会其中的细节。

1. 求最小幂 2^k

C++语言：

```
int Lowbit(int t)
{
    return t & ( t ^ ( t - 1 ) );
}                              //end
```

Pascal 语言：

```
Function lowbit(x:longint):longint;
var  t:longint;
begin
  t:=x and (x (xor(x-1));
  exit(t);     //相当于 lowbit:=t;
end;
```

建立树状数组c、更新元素值、子序列求和，都与 2^k有关。所以令lowbit(i)=2^k。（其中k为i在二进制下末尾 0 的个数）

其实lowbit(i)=2^k=i and (i xor (i-1))

以 i=6 为例

$$
\begin{array}{ll}
 & (6)_{10}=(0110)_2 \\
\text{Xor} & \underline{(6-1)_{10}=(5)_{10}=(0101)_2} \\
 & (0011)_2 \\
\text{And} & \underline{(6)_{10}=(0110)_2} \\
 & (0010)_2
\end{array}
$$

2. 对某个元素进行加法操作：将 a[p]的值加上一个值 x(x 可正可负)

C++语言：

```
int plus(int p , int x)
{
    while(p <= maxn)
        //maxn 为给定范围的上限
    {
        c[p] += x;
        p += Lowbit(p);
    }
}
```

Pascal 语言：

```
procedure plus(p,x:longint);
begin
  while p<=maxn do
            //maxn 为给定范围的上限
  begin
      c[p]:=c[p]+x;
      p:=p+lowbit(p)
  end;
end;
```

譬如，修改 9 这个值，我们要修改

c[9] = c[1001]

c[9+2^0] = c[10] = c[1010]

c[10+2^1] = c[12] = c[1100]

c[12+2^2] = c[16] = c[10000]

………

修改 12 这个值，我们要修改

c[12] = c[1100]

c[12+2^2] = c[16] = c[10000]

c[16+2^4] = c[32] = c[100000]

依此类推，直到上限

3. 求前 p 项和：统计 a[1]...a[p]的和

C++语言：

```
int sum(int p)
{
    int sum = 0;
    while(p > 0)
    {
        sum+ = c[p];
        p- = Lowbit(p);
    }
    return sum;
}
```

Pascal 语言：

```
Function sum(p:longint):longint;
var
    s:longint;
begin
  s:=0;
  while p>0 do
  begin
    s:=s+c[p];
    p:=p-lowbit(p);
  end;
  exit(s);      //相当于 sum:=s;
end;
```

譬如，求 a[1]+a[2]+a[3]+…+a[12]的和

c[12] = c[1100]

c[12-2^2] = c[8] = c[1000]

c[8-2^3] = c[0]

所以 a[1]+a[2]+a[3]+…+a[12]的和为 c[12]+c[8]

4. 统计 a[x]..a[y]的值

调用以上的 sum 操作：sum[y]-sum[x-1]

一维的树状数组的每个操作的复杂度都是 O(logn)的，非常高效。它可以扩充为 n 维，这样每个操作的复杂度就变成了 O((logn)^n)，在 n 不大的时候可以接受。扩充的方法就是将原来改变和查询的函数中的一个循环改成嵌套的 n 个循环在 n 维的 c 数组中操作。

要注意树状数组能处理的是下标为 1..n 的数组，绝对不能出现下标为 0 的情况。因为 lowbit(0)=0，这样会陷入死循环。

实战运用

【例 1】给定 n 个数列，规定有两种操作，一是修改某个元素，二是求子数列[a,b]的连续和。数列的元素个数最多 10 万个，询问操作最多 10 万次。

【输入格式】

第一行2个整数n,m（n表示输入n个数列，m表示有m个操作）

第二行输入n个数列。

接下来M行，每行有三个数k,a,b（k=0表示求子数列[a,b]的和，k=1表示第a个数列加b值）。

【输出格式】

输出若干行数字，表示每次K=0时对应输出一个子数列[a,b]的连续和。

【输入样例】

```
10  5                     //输入 n 个数列，有 m 个操作
1 2 3 4 5 6 7 8 9 10      //输入 n 个数
1 1 5                     //第 1 个数加上 5
0 1 3                     //求数列 1 到数列 3 的连续和为 11
0 4 8                     //求数列 4 到数列 8 的连续和为 30
1 7 5                     //第 7 个数加上 5
0 4 8                     //求数列 4 到数列 8 的连续和为 35
```

【输出样例】

```
11
30
35
```

【参考程序】

```
Program treearray;
const maxn=1000;
var c:array[1..maxn]of longint;
    n,i,m,x,y,st,a,b:longint;

function lowbit(x:longint):longint;
begin
  lowbit:=((x-1)xor x)and x;
end;
```

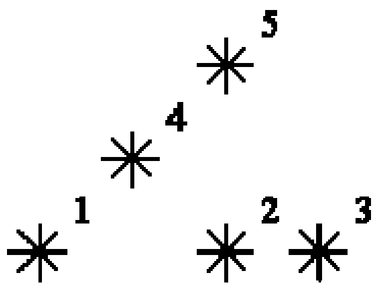
```
procedure plus(p,x:longint);
begin
  while p<=n do
   begin
     inc(c[p],x);
     inc(p,lowbit(p));
   end;
end;

function sum(p:longint):longint;
var s:longint;
begin
  s:=0;
  while p>0 do
    begin
       inc(s,c[p]);
       dec(p,lowbit(p));
    end;
  exit(s);
end;

Begin
  readln(n,m);
  fillchar(c,sizeof(c),0);
  for i:=1 to n do
    begin
      read(y);
      plus(i,y);
    end;
  for i:=1 to m do
   begin
     readln(st,a,b);
     if st=0 then writeln(sum(b)-sum(a-1))
             else plus(a,b);
   end;
End.
```

【例2】数星星Stars(ural1028)

【问题描述】

天空中有一些星星，这些星星都在不同的位置，每个星星有个坐标。

如果一个星星的左下方（包含正左和正下）有k颗星星，就说这颗星星是k级的。



比如，在上面的例图中，星星5是3级的（1，2，4在它左下）。

星星2，4是1级的。例图中有1个0级，2个1级，1个2级，1个3级的星。

【编程任务】

给定星星的位置，输出各级星星的数目。

给定n个点，定义每个点的等级是在该点左下方（含相等）的点的数目，试统计每个等级有多少个点。（n<=15000，0<=x,y<=32000）

【输入格式】

第1行一个整数N(1<=N<=15000)，表示星星的数目。

接下来N行给出每颗星星的坐标，两个整数X,Y(0<=X,Y<=32000)

不会有星星重叠。星星按Y坐标增序列出，Y坐标相同的按X坐标增序列出。

【输出格式】

N行，每行一个整数，分别是0级，1级，2级……N-1级的星星的数目。

【输入样例】

5
1 1
5 1
7 1
3 3
5 5

【输出样例】

1
2
1
1
0

【算法分析】

相当经典的树状数组题目，一开始分析题目是第一感觉是二维的树状数组，不过数据范围显然不容许的。对于每个星星按y坐标从小到大排序，相同y坐标按x坐标从小到大排序（题目中数据已经有序），输入顺序已排好序，那么只要依次统计星星i之前x坐标小于等于i.x的星星有多少，即是星星i的级别。y坐标没用，显然是一维树状数组应用，这样也就成了树状数组的模型，编码很简单。

设数组a[x]初始为0，表示在x坐标处星星的个数，则求和a[0]~a[x]则为该星星的等级，逐个处理每个星星。

有一个地方尤其要注意，树状数组是以1号为起始的，而且只能用1号。x可能为0，为0会时陷入死循环，处理时要将所有的x+1。(当然加上其它的也无所谓，只是上限范围需要变大)，还有就是x的范围不能事先确定，在plus的时候我直接加到了x取值范围的最大值。

【例3】校门外的树(vijo1448)

【问题描述】

校门外有很多树，有苹果树，香蕉树，有会扔石头的，有可以吃掉补充体力的□□

如今学校决定在某个时刻在某一段种上一种树，保证任一时刻不会出现两段相同种类的树，现有两个操作：

K=1，读入l,r表示在l - r之间种上的一种树

K=2，读入l,r表示询问l - r之间能见到多少种树（l,r>0,道路总长和操作数<=50000）

【输入文件】

第一行 n,m 表示道路总长为 n，共有 m 个操作

接下来 m 行为 m 个操作。

【输出文件】

对于每个 k=2 输出一个答案

【输入输出样例】

tree.in	tree.out
5 4 1 1 3 2 2 5 1 2 4 2 3 5	1 2

【算法分析】

这题看起来和树状数组没什么关系，不过我们通过一定的转化，可以利用树状数组很好地解决这个问题。

我们不妨把所有线段的端点看成括号序列，即把询问的区间[lq,rq]看成在横坐标lq处的一个‘[’和rq处的‘]’，即把插入的线段[li,ri]看成在横坐标li处的一个‘(’和ri处的‘)’。

稍作分析，我们不难发现，最后的答案等于‘]’左边‘(’的个数减去‘[’左边‘)’的个数。

那么我们现在做的就是对某个点做修改，对某个前缀求和。我们就可以很容易想到树状数组的做法：

1.建立两个树状数T1和T2，分别维护‘(’和‘)’。

2.若K=1，读入li,ri，plusT1(li,1)，plusT2(ri,1)。

3.若K=2，读入lq,rq，sumT1(rq-1)-sumT2(lq-1)

小结

经过前面概念和例题的讨论，我们应该体会到了树状数组的特点：程序短、速度快；也明白到它最常用的功能：维护部分和。在竞赛中，如果我们采用树状数组处理，既可以保证程序的效率，也可以大幅度减小调试难度，为选手省下大量的时间，可以说是极其划算的。

总结

1、树状数组是和线段树有异曲同工的数据结构，其优点在于高效，写起来很好写，并且是在线的数据结构。

2、但其只能处理一次修改某一个点上的数据，并不能有效处理一次修改一个区间的数据（每次只能修改某一节点的值而不能修改整个树{可以理解为区间}的值）。

3、能用树状数组的题，一定能用线段树做，但能用线段树做的题，不一定能用树状数组做。