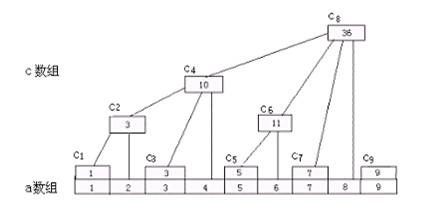
**(ZZ)Binary indexed tree-树状数组**

[Algorithm](http://mcfaddan.blog.163.com/blog/#m=0&t=1&c=fks_087074092084083068092082087095085082081075085081081066086) 2010-04-13 15:46:50 阅读15 评论0   字号：大中小 订阅

from <http://duanple.blog.163.com/>

虽然网上有很多介绍了，我还是要写一下吧，尽量从它的起源，如何被发现，以及为什么应该是这样的来写，单纯的使用很简单，也不是学习的目的，理解有助于记忆吧，当然可能还有理解不到的地方。  
  
起源  
  
树状数组英文名称是Binary Indexed Trees，最早由Peter M. Fenwick于1994年MARCH以A New Data Structure for Cumulative Frequency Tables为题发表在SOFTWARE PRACTICE AND EXPERIENCE。最初是为了解决数据压缩里的累积频率(Cumulative Frequency)的计算问题提出来的。再此之前采用的方法在这篇论文里都有提到比如MTF(move-to-front),HEAP,SPLAY。现在则因为实现简单，结构单一，作为计算前缀和的在线数据结构被广泛采用。  
  
按照Peter M. Fenwick的说法，该想法产生大概类比了整数与二进制的关系.  
Each integer can be represented as sum of powers of two. In the same way, cumulative frequency can be represented as sum of sets of subfrequencies. In our case, each set contains some successive number of non-overlapping frequencies.  
也就是说就像所有的整数都可以表示成2的次方的和那样，我们也可以考虑把一串序列表示成一系列子序列的和。而实际上也正是采用这个想法，将一个前缀和划分成了多个子序列的和，而划分的方法与数的2的幂和具有及其相似的方式。首先子序列的个数也是其二进制表示中1的个数，同时子序列代表的f[i]的个数也是2的幂，这些与Integer都很类似。  
  
之所以命名为Binary indexed tree，Peter M. Fenwick有这样的一段解释，  
In recognition of the close relationship between the tree traversal algorithms and the binary representation of an element index,the name "binar indexed tree" is proposed for the new structure.  
  
前缀和的拆分：  
  
比如我们假设用C[i]表示f[1]...f[i]的和，而用tree[idx]表示子序列，按照定义实际上tree[idx]是那些indexes from (idx - 2^r + 1) to idx的f[index]的和,其中r是idx最右边的那个非零位到右边末尾的0的个数。也就是说实际上tree[idx]的子序列中f[index]的个数也是2的幂，这点与integer也类似。通过对tree[idx]的这种定义，便可以方便的把C[idx]用若干个tree[idx]表示出来。实际上这样C[idx]可以类比成某个Integer，而tree[idx]的f[i]个数可以类比成用来组合成该Integer的2的幂。  
举个例子：比如我们想求C[13]也就是f[1]+...+f[13];则C[13] = tree[13]+tree[12]+tree[8],另：  
tree[13] = f[13]  
tree[12] = f[12]+f[11]+f[10]+f[9]  
tree[8] = f[8]+....f[1]  
对应于Integer 13 = 1+4+8,上面tree[13],tree[12],tree[8]的f个数刚好也是1 4 8,完全统一。  
  
{--------------------------------------------------------------------------------------------------------------------------------------  
来一个引用自别人的图和图的解释：  
[](http://img.blog.163.com/photo/LUC3ybJmk2A8A1V_oY4cnA==/3424987516616419862.jpg)  
令这棵树的结点编号为C1，C2...Cn。令每个结点的值为这棵树的值的总和，那么容易发现：  
C1 = A1  
C2 = A1 + A2  
C3 = A3  
C4 = A1 + A2 + A3 + A4  
C5 = A5  
C6 = A5 + A6  
C7 = A7  
C8 = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8  
...  
C16 = A1 + A2 + A3 + A4 + A5 + A6 + A7 + A8 + A9 + A10 + A11 + A12 + A13 + A14 + A15 + A16  
  
这里有一个有趣的性质，下午推了一下发现：  
设节点编号为x，那么这个节点管辖的区间为2^k（其中k为x二进制末尾0的个数）个元素。因为这个区间最后一个元素必然为Ax，  
所以很明显：Cn = A(n – 2^k + 1) + ... + An  
-----------------------------------------------------------------------------------------------------------------------------------------}  
  
前缀和的计算：  
  
姑且这样猜测一下，我想Peter M. Fenwick当初也是最先根据这个猛然间的类比，做出了这样的分解的尝试，逐步把前缀和分解为部分和，写出了上面的表示方式。因为如果可以分解成这种方式，就意味着如果所有的tree[idx]可以得到，那么任意的C[i]就可以在lgn时间内算出了。而对于其他属性的发现，则是这样的尝试之后必然存在的，不过他发现了它们。也就是文章本天成，妙手偶得之。或者说it exist in The Book.  
  
更新的维护：  
  
然后继续考虑，当某个f[i]改变的时候，如何才能维护该f[i]所涉及到的tree[idx]就可以了，也就是找到那些包含了f[i]的tree[idx]元素，只要更新它们就可以了。  
  
然后初步观察，可以发现每个f[i]所隶属的tree[idx]的数目实际上不会超过lgn，下面的任务只有找出这些tree[idx]就可以了。这样基本上就完成了这个数据结构所应该做的工作。为什么每个f[i]所隶属的tree[idx]不会超过lgn呢？  
  
虽然不是显而易见的，但也不是很隐蔽的。对于每个tree[idx] = f[idx]+f[idx-1]....+f[idx-2^r+1]，如果f[i]属于tree[idx]中的一员，应该满足 idx-2^r+1  =< i <= idx,看起来还不是很明显，也就是说idx满足了这个条件，就可以包含i了，所以如果求包含i的tree[idx]的数目，实际上是求满足这个不等式的idx的个数。  
  
对于这个条件来说，转换成更明显的文字来说，就是idx应当具有如下性质在剪掉末尾1之前，它应当大于等于i，剪掉末尾1后，要小于等于i。对一系列的idx：a1000..,如果要使i处在a1000和a0000之间需要满足如下条件，即a0000 <= i <= a1000,则a应当是固定的了，我们的idx具有这样的特征它的左半段a跟我们的i是相同的，剩下的就是1000了，因此1的位置只有Lgn种可能，而且只有该位置的二进制表示为0时，才可能存在对应的idx。有一点需要注意的是当a1000...长度大于i时，是一种特殊情况，可以单独讨论，当然也可以把i的前导0补足，如下例。  
  
以5(00101)为例，则包含它的idx有6(00110) 8(01000) 16(10000)这些，观察6 8 16可以发现，它们实际上就是把5的某个非0位变成了1，同时把它右面的所有二进制位都变成了0。而在进一步的探索过程中，实际上这样的一个过程便是通过不断的i=i+lowbit(i),来实现的。  
  
实现  
需要支持的操作有：  
计算某个前缀和：Read cumulative frequency  
更新某个元素f[i]的值：Change frequency at some position and update tree  
读取某个位置的f值：Read the actual frequency at a position  
缩放整个数组，比如数值全部减半：Scaling the entire tree by a constant factor  
找到具有给定值的f[i]：Find index with given cumulative frequency  
  
最常用的前三个，后两个均可以增加一部分存储f[i]元素的空间实现。  
进行这些操作之前有一个必备的基本运算，lowbit,即计算整数里最右边的那个非零位，在Peter M. Fenwick的论文中提到了三种方法。

   return x-(x&(x–1));

   return x&(-x);

   return x&(2^k-x);2^k is a power of 2 greater than the table size.

如果要统计f[1]到f[idx]之间的和，可以通过调用如下函数实现

int read(int idx){  
 int sum = 0;  
 while (idx > 0){  
 sum += tree[idx];  
 idx -= (idx & -idx);  
 }  
 return sum;  
}

如果要把f[idx]增加var，可以通过调用如下函数实现

void update(int idx ,int val){  
 while (idx <= MaxVal){  
 tree[idx] += val;  
 idx += (idx & -idx);  
 }  
}

读取某个位置的值f[i]：  
一般我们可以用sum[i]-sum[i-1]，根据此结构还可以继续优化。即计算两个sum的时候，它们的路径可能会相遇。

int readSingle(int idx){  
int sum = tree[idx]; *// sum will be decreased*  
if (idx > 0){ *// special case*  
 int z = idx - (idx & -idx); *// make z first*  
 idx--; *// idx is no important any more, so instead y, you can use idx*  
 while (idx != z){ *// at some iteration idx (y) will become z*  
 sum -= tree[idx];   
*// substruct tree frequency which is between y and "the same path"*  
 idx -= (idx & -idx);  
 }  
}  
return sum;  
}  
  
树的所有元素\*一个常数因子，可以通过两种方式实现  
void scale(int c){  
 for (int i = 1 ; i <= MaxVal ; i++)  
 update(-(c - 1) \* readSingle(i) / c , i);  
}  
void scale(int c){  
 for (int i = 1 ; i <= MaxVal ; i++)  
 tree[i] = tree[i] / c;  
}  
寻找具有给定值的tree[i],最简单的方法当然还是将所有的tree[i]计算一遍，当然对于包含负数的数组，我们只能这样做  
但是对于，那些单调的tree[i]序列来说，完全可以采用二分查找的方法。  
*// if in tree exists more than one index with a same  
// cumulative frequency, this procedure will return   
// some of them (we do not know which one)  
  
// bitMask - initialy, it is the greatest bit of MaxVal  
// bitMask store interval which should be searched*  
int find(int cumFre){  
 int idx = 0; *// this var is result of function*  
   
 while ((bitMask != 0) && (idx < MaxVal)){ *// nobody likes overflow :)*  
 int tIdx = idx + bitMask; *// we make midpoint of interval*  
 if (cumFre == tree[tIdx]) *// if it is equal, we just return idx*  
 return tIdx;  
 else if (cumFre > tree[tIdx]){   
 *// if tree frequency "can fit" into cumFre,  
 // then include it*  
 idx = tIdx; *// update index*   
 cumFre -= tree[tIdx]; *// set frequency for next loop*   
 }  
 bitMask >>= 1; *// half current interval*  
 }  
 if (cumFre != 0) *// maybe given cumulative frequency doesn't exist*  
 return -1;  
 else  
 return idx;  
}  
  
  
  
*// if in tree exists more than one index with a same  
// cumulative frequency, this procedure will return   
// the greatest one*  
int findG(int cumFre){  
 int idx = 0;  
   
 while ((bitMask != 0) && (idx < MaxVal)){  
 int tIdx = idx + bitMask;  
 if (cumFre >= tree[tIdx]){   
 *// if current cumulative frequency is equal to cumFre,   
 // we are still looking for higher index (if exists)*  
 idx = tIdx;  
 cumFre -= tree[tIdx];  
 }  
 bitMask >>= 1;  
 }  
 if (cumFre != 0)  
 return -1;  
 else  
 return idx;  
}

总结  
BIT很容易编码实现  
所有的查询均花费logn或者常数时间  
需要线性数量级的空间  
可以扩展到n维的情况，BIT可以作为一种多维的数据结构，即可扩展到多维。以上只是一维的情况。  
  
附：

POJ上用到树状数组的题目：

[poj2352 Stars](http://acm.pku.edu.cn/JudgeOnline/problem?id=2352)  
[poj2481 Cows](http://acm.pku.edu.cn/JudgeOnline/problem?id=2481)  
[poj1195 Mobile phones](http://acm.pku.edu.cn/JudgeOnline/problem?id=1195)   
[poj3067 Japan](http://acm.pku.edu.cn/JudgeOnline/problem?id=3067)

[poj2155 Matrix](http://acm.pku.edu.cn/JudgeOnline/problem?id=2155)  
[poj2464 Brownie Points II](http://acm.pku.edu.cn/JudgeOnline/problem?id=2464)

[poj3321 apple tree](http://acm.pku.edu.cn/JudgeOnline/problem?id=3321)

Poj3321 [Apple Tree](http://acm.pku.edu.cn/JudgeOnline/problem?id=3321) 树状数组。注意需要首先构图，即把每次查询转换为对于数组中某个子序列和的查询。这样每个节点实际上代表了一系列的节点之和，需要把节点重新编号，使得子序列变成连续的。就可以运用树状数组了。

Poj2155 [Matrix](http://acm.pku.edu.cn/JudgeOnline/problem?id=2155) 二维树状数组。注意一个转化，即把更新与求和对应的down,up互换.即考虑一维时，比如需要更新[a,b],我们只要更新b的down集合+1，以及a-1的down集合-1就可以，这样就可以保证只更新了[a,b]，因为他们中的节点的父亲集合里只有一个会被更新。而查询时只需要计算所有父亲的和就可以得到被修改的次数了。然后一维的情况便可以推广到二维情况里，即[x1,y1][x2,y2]组成的矩形，一次置0可以分解成对四个以原点[1,1]为左上点的矩形的运算和。即把一个矩形的操作，转化成4个矩形的操作。  
  
参考资料：  
[A New Data Structure for Cumulative Frequency Tables](http://www.cs.ubc.ca/local/reading/proceedings/spe91-95/spe/vol24/issue3/spe884.pdf)  
[Topcoder - Binary Indexed Trees](http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=binaryIndexedTrees)  
[树状数组](http://hi.baidu.com/wuxyy/blog/item/a4ad808b59be8bd2fd1f109d.html)  
[Binary Indexed Tree](http://www.nocow.cn/index.php/%E6%A0%91%E7%8A%B6%E6%95%B0%E7%BB%84)