文章编号:1005-0523(2011)02-0045-05

基于树状数组的逆序数计算方法

周 娟[1],曹义亲[1],谢 昕[2]

(华东交通大学1. 软件学院; 2. 信息学院,江西 南昌 330013)

摘要:n个元素组成的置换$a[1],a[2],\cdots,a[n]$。若$i<j$且$a[i]>a[j]$,则称$(a[i],a[j])$是一个逆序对。置换中逆序对的个数称为置换的逆序数。按定义,计算逆序数要通过$n(n-1)/2$此次比较,时间复杂度是$O(n^2)$。设计了一种新的方法,利用树状数组计算逆序数,时间复杂度降为$O(n\log_2(n))$。主要思路是将元素从大到小依次放置在数状数组中,对于每一个元素i来说,因它前面的数比它大而计算出逆序数$t[i]$,利用树状数组的结构特征,即可以$O(\log_2(n))$的时间复杂度而统计出$t[i]$,那么最终总的逆序数为$\sum t[i]$。

关键词:树状数组;逆序数;置换;算法;线段树

中图分类号:O241.6;TP301.6 **文献标识码**:A

n个数的置换$a[1],a[2],\cdots,a[n]$也称为排列或数组[1-2],若$i<j$且$a[i]>a[j]$,则称$(a[i],a[j])$是一个逆序对。置换中的逆序对的个数称为置换的逆序数[1,3]。逆序数是奇数的置换叫奇置换,否则叫偶置换。逆序数一般分成n个部分进行计算。令$t[i]$表示逆序对$(a[j],i)$的个数,即排在i的左边且比i大的数的个数,则逆序数为$t[1]+t[2]+\cdots+t[n]$。文[4]中利用轮换和归并排序计算奇偶性。一般来说,$t[i]$的计算方法是:设置1个全0的排列$c[1],c[2],\cdots,c[n]$,对大于i的数$a[j]$,将$c[j]$置为1,设i在位置p,计算位置p左边1的个数就是$t[i]$,然后置$c[p]=1$,下一步计算$t[i-1]$。

例1 设$n=8$,$a=\{3,2,1,5,8,4,6,7\}$,则$t[1]=2$,$t[2]=1$,$t[3]=0$,$t[4]=2$,$t[5]=0$,$t[6]=t[7]=1$,$t[8]=0$。上述方法的时间复杂度是n^2。本文提出一种复杂度为$n\log_2 n$的计算逆序数的算法,采用树状数组[5]。

1 树状数组

定义1 设i整除2^k,i不能整除2^{k+1},k为非负整数,则称2^k是i的最小比特,记为lowbit(i)。

例2 8的最小比特是8,6的最小比特是2。把i化为2进制数后,最后的连续的0的个数就是k。奇数的最小比特是1,若$i=2^k$,则i的最小比特就是i。

定义2 设$c[1],c[2],\cdots,c[n]$是一个数组,$c[i]$称为点或数。定义$c[i]$的父节点是$c[i+\text{lowbit}(i)]$。称c为树状数组。

性质1 若i是奇数,则$c[i]$没有子节点。以$c[i]$为根的子树只有1个点。

性质2 若i的最小比特是2^k,则以$c[i]$为根的子树有2^k个点,有k个子节点,它的k个子节点是(若i的二进制数是***100000,以k=5为例):

c[***10000],c[***11000], c[***11100], c[***11110], c[***11111]

例3 $c[8]$为根的子树有8个点,$c[8]$的子节点是$c[4]$,$c[6]$,$c[7]$。$c[4]$的子节点是$c[2]$,$c[3]$。

收稿日期:2010-12-30

基金项目:国家自然科学基金项目(11061014);江西省自然科学基金项目(2010GZS0031);华东交通大学科学研究项目(09RJ07)

作者简介:周 娟(1977—),女,讲师,研究方向为算法设计与分析。

c[6]的子节点是c[5]。如表1和图1所示。

表1 树状数组

Tab.1 Arborescence array

i	lowbit(i)	i的父节点i+lowbit(i)
1	1	2
2	2	4
3	1	4
4	4	8
5	1	6
6	2	8
7	1	8
8	8	16

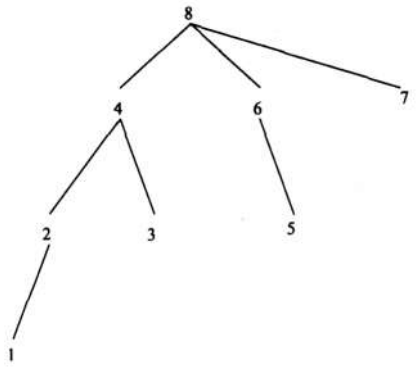

图1 树状数组-空树

Fig.1 Empty arborescence array

图1是一颗空树，尚未放入需要计算的项目，依定义2即可得到此父子结构的一颗确定的树，据此结构特征，再通过定义c的内涵以及具体问题所引入的数组b的内涵等，即可用来解决具体计算问题。也就是说，树状数组是一个特定结构的有利工具。

定义3 设c[1]，c[2]，…，c[n]是树状数组，b[1]，b[2]，…，b[n]是与之对应的一个n元数组。

j=lowbit(i)，定义c[i]的值为

$$c[i]= b[i-j+1]+ b[i-j+2]+\cdots+b[i]。 \tag{1}$$

即c[i]是到i为止的lowbit(i)个数的和。

例4 设n=10，则

c[1]=b[1]，　c[2]=b[1]+b[2]，

c[3]=b[3]，　c[4]=b[1]+b[2]+b[3]+b[4] =c[2]+b[3]+b[4]，

c[5]=b[5]，　c[6]=b[5]+b[6]，

c[7]=b[7]，　c[8]= b[1]+b[2]+b[3]+b[4]+ b[5]+b[6]+ b[7]+b[8]=c[4]+c[6]+c[7]+b[8]，

c[9]=b[9]，　c[10]=b[9]+b[10]。

由例4可以看出c[i]是b[i]与c[i]的子节点的值之和，如图2所示。

算法1 计算i的最小比特2^k：

```
int lowbit(int i)
{
    return  i&(i^(i-1));// &按位与和^按位异或是c语言的位运算符[6-7])
}
```

算法2 计算前m项和：

```
int Sum(int  m )
{
    int sum=0;
    while(m>0)
    {  sum+=c[m];
       m-=lowbit(m);
    }
    return sum;
```

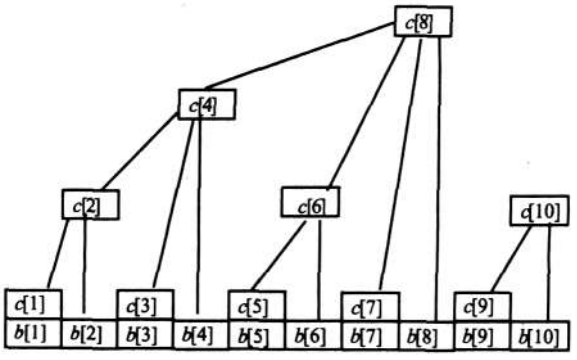

图2 树状数组c和数组b

Fig.2 Arborescence array c and array b

```
}
```

由此可知,若m的二进制数中有s个1,则计算前m项的和只要计算s次,即计算$t[i]$的次数为s($\leqslant k$)小于$\log_2(n)$次。那么总次数小于$n\log_2(n)$

算法3 如果某个数值发生改变,例如$b[p]$的值增加了num,那么$c[p]$和$c[p]$的父节点,以及其父节点的父节点……的值都要增加num:

```
void plus(int p,int num)
{
    while(p<=n)
    {
      c[p]+=num;
      p=p+lowbit(p)
     }
}
```

2　树状数组计算逆序数

算法4 计算置换$a[1],a[2],\cdots,a[n]$(是1,2,…,n的一个排列)的逆序数Num =$t[1]+t[2]+\cdots+t[n]$:

```
Num=0;
for(i=1;i<=n;i++) at[a[i]]=i;  // 引入数组 at, at[i]表示 i在第at[i]个位置
for(i=1;i<=n;i++) c[i]=0;
for(x=n;x>=1;x--)
{
    p=at[x];// 把数x 放在第p个位置
    Num+=Sum(p);//位置p左边有多少个数(比x大的数)
    plus(p,1);// 更新 c[p]和c[p]的祖先的值
}
```

下面以例1来诠析算法4。

1) 首先建立一棵空树(树状数组),如图1,这棵树共有8个位置(或称节点),每个位置上将放置数组a的一个元素,放置完毕如图3;$at[a[i]]$表示元素$a[i]$放置在$at[a[i]]$节点上。

2) **定义4**　b为节点上所放元素的个数,对于图1,$b[i]=0,i=1,\cdots,8$;对于图2,$b[i]=1,i=1,\cdots,8$。

3) 定义数组c,$c[i]$为以节点i为根的树上所含元素的个数,满足式(1)。

4) **定义**　sum(i)为放置在i之前的节点,所含元素的个数和,可以运用算法2。

5) 首先放置最大的元素a_max,$a\_max=max(a[1],\cdots,a[n])$,即$n$;将它放置在题设所放的位置,$at[a_max]$处,对于最大数的逆序数$t[at[a_max]]$为0,因为在它之前没有比它大的数,此时位置$at[a_max]$的左边也是空的,如图4中(a)第1步;再放第2大的数,如果它放在a_max之前,那么它的逆序数也为0,否则为1,如图4中(b)第2步;依次放置,某个数x放置在$at[x]$处,此时求x的逆序数$t[at[x]]$,即为求在$at[x]$位置的左下方和下方一个有多少个数已经放置了,即为sum($at[x]$);每放置一个元素,就以plus来更新c,计算过程和树状数组的维护过程如表2和图4。

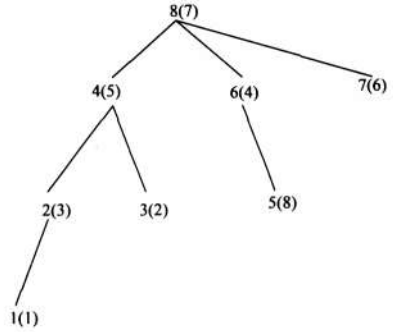

图3　树状数组-放入置换后(1,3,2,5,8,4,6,7)依各元素位置放入树中
Fig.2　Add array to the arborescence array with 1,3,2,5,8,4,6,7 in turn

表2 例1的计算过程

Tab.2 Calculation process for example 1

i, $at[i]$	$x=a[i]$	第1步 放8于5		第2步 放7于8		第3步 放6于7		第4步 放5于4		第5步 放4于6		第6步 放3于1		第7步 放2于2		第8步 放1于3	
		$c[i]$	$t[x]$	$c[i]$	$t[x]$	$c[i]$	$t[x]$	$c[i]$	$t[x]$	$c[i]$	$t[x]$	$c[i]$	$t[x]$	$c[i]$	$t[x]$	$c[i]$	$t[x]$
1	3	0	0	0	0	0	0	0	0	0	0	0→1	0	1	0	1	0
2	2	0	0	0	0	0	0	0	0	0	0	0→1	0	1→2	0→1	2	1
3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0→1	0→2
4	5	0	0	0	0	0	0	0→1	0	1	0	1→2	0	2→3	0	3→4	0
5	8	0→1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
6	4	0→1	0	1	0	1	0	1	0	1→2	0→2	2	2	2	2	2	2
7	6	0	0	0	0	0→1	0→1	1	1	1	1	1	1	1	1	1	1
8	7	0→1	0	1→2	0→1	2→3	1	3→4	1	4→5	1	5→6	1	6→7	1	7→8	1

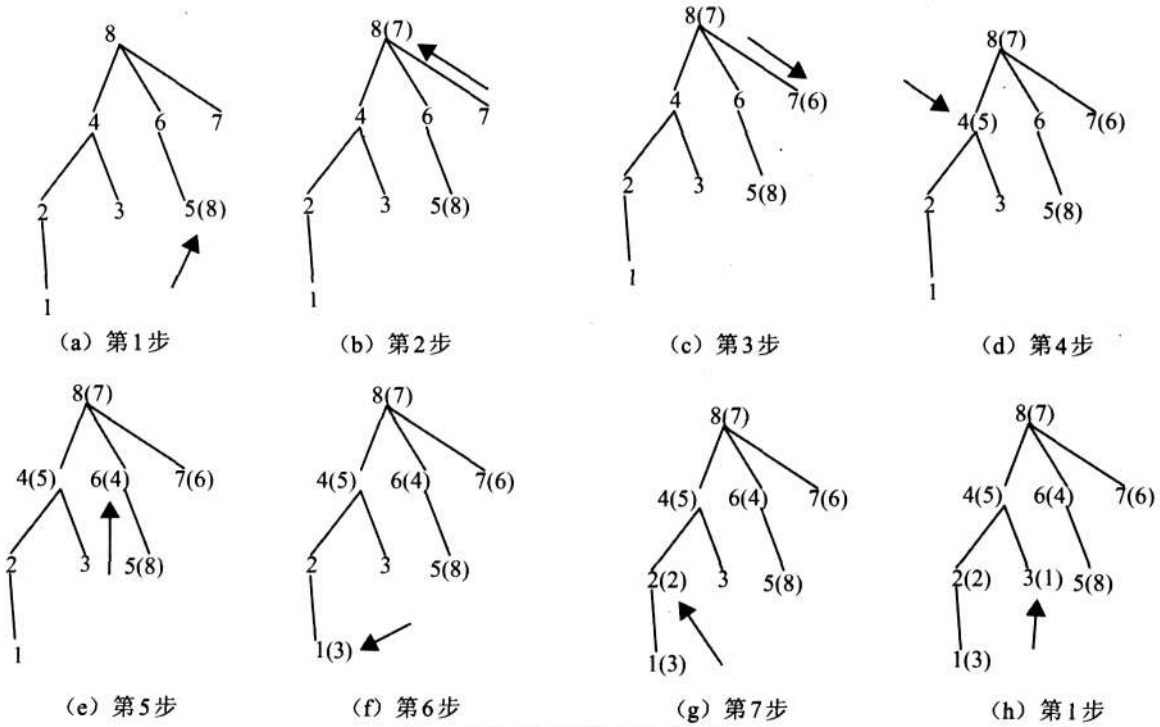

图4 例1的树状数组维护过程

Fig.4 Maintenance process of arborescence array for example 1

3 总结

树状数组是一个查询和修改复杂度都为log(n)的数据结构,并支持随时修改某个元素的值,复杂度也为log(n)。它可以很高效地进行区间统计。在思想上类似于线段树[8-10],比线段树节省空间,编程复杂度比线段树低,但适用范围比线段树小。总之树状数组有着简单易用的特点,容易记忆。

参考文献:

[1] RICHARD A BRUALDI. 组合数学[M]. 5版. 北京:机械工业出版社,2009:54-55.

[2] 王延明. 关于排列逆序数的进一步性质及其应用[J]. 枣庄学院学报,2007,24(5):12-13.

[3] 张翠, 张英瑞.关于逆序数相同的n级排列个数的讨论[J]. 洛阳师范学院学报,2010,29(5):24-25.

[4] 周尚超. 置换奇偶性的快速算法[J]. 华东交通大学学报,2007,24(1):87-89.

[5] 刘洋. 基于纹理合成的图像修复与基于分形的图像分割方法的研究与应用[D]. 吉林:吉林大学,2010:71-73.

[6] 谭浩强. C程序设计教程[M].北京:清华大学出版社,2007:33.

[7] 钱能. C++程序设计教程[M]. 2版. 北京:清华大学出版社,2005:119-122.

[8] 林盛华. 线段树在程序设计中的应用[J]. 大众科技,2005(4):68-69.

[9] 李博涵, 郝忠孝.近似查询中重叠区域的扫描计算[J], 计算机工程,2008,34(13):10-12.

[10] CHANG YEIN, WU CHENCHANG, SHEN JUNHONG, et al. Range queries based on a structured segment tree in p2p systems[C]//Proceedings of the 2010 17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems,2010:91-99.

Algorithm of Inverse Number Based on Arborescence Array

Zhou Juan[1], Cao Yiqin[1], Xie Xin[2]

(1. School of Software; 2. School of Information Engineering, East China Jiaotong University, Nanchang 330013, China)

Abstract: Suppose there is a permutation of $a[1],a[2],\cdots,a[n]$ that is constituted by n elements.If $i<j$ and a $[i]>a[j]$, then $(a[i],a[j])$ is called an inverted sequence pair. The number of inverted sequence in a permutation is called inverse number of a permutation. According to the definition, the caculation of inverse number must be compared with $n(n\text{-}1)/2$, and the time complexity is $O(n^2)$. A new method is devised to calculate the reverse numble by arborescence array and reduce the time complexity to $O(n\log_2(n))$.The principal idea of this method is to place elements into arborescence array in descending order. With regard to each element, since the previous number which is bigger and being worked out at reverse number $t[i]$, when taking advantage of the architectural feature of arborescence array, it will come to $t[i]$ by the time complexity of $O(\log_2(n))$ and finally reach the total reverse number as $\sum t[i]$.

Key words: arborescence array ; inverse number; permutation;algorithm; segment tree