树状数组求逆序数 收藏

逆序数就是数中各位在它前面有多少个数比它大，求出这些元素个数之和。

今天看了个树状数组，可以很好的解决这个问题，普通方法需要O(N^2)复杂度,用树状数组只需要O(NlongN)

树状数组实际上还是一个数组，只不过它的每个元素保存了跟原来数组的一些元素相关的结合值。

若A为原数组，定义数组C为树状数组。C数组中元素C[ i ]表示A[ i –lowbit( i ) + 1]至A[ i ]的结合值。

lowbit(i)是i的二进制中最后一个不为零的位数的2次方，可以这样计算

lowbit(i)=x&(-x)

lowbit(i)=x&(x^(x-1))

当想要查询一个sum(n)时，可以依据如下算法即可：

step1:　令sum = 0，转第二步；

step2:　假如n <= 0，算法结束，返回sum值，否则sum = sum + Cn，转第三步；

step3: 令n = n – lowbit(n)，转第二步。

n = n – lowbit(n)这一步实际上等价于将n的二进制的最后一个1减去。而n的二进制里最多有log(n)个1，所以查询效率是log(n)的。

修改一个节点，必须修改其所有祖先，最坏情况下为修改第一个元素，最多有log(n)的祖先。所以修改算法如下（给某个结点i加上x）：

step1: 当i > n时，算法结束，否则转第二步；

step2: Ci = Ci + x， i = i + lowbit(i)转第一步。

i = i +lowbit(i)这个过程实际上也只是一个把末尾1补为0的过程。

求逆序的思路：

可以把数一个个插入到树状数组中， 每插入一个数， 统计比他小的数的个数，对应的逆序为 i- getsum( data[i] )，其中 i 为当前已经插入的数的个数， getsum( data[i] ）为比 data[i] 小的数的个数，i- getsum( data[i] ) 即比 data[i] 大的个数， 即逆序的个数。最后需要把所有逆序数求和，就是在插入的过程中边插入边求和。

下面是代码：

view plaincopy to clipboardprint?

#include <iostream>

using namespace std;

#define N 10

struct Node{

int data;

int pos;

};

Node d[N+1];

int inverse[N+1];

int count[N];

int cmp(const void\*a,const void\*b)

{

Node \*pa=(Node\*)a;

Node \*pb=(Node\*)b;

return pa->data-pb->data;

}

int lowbit(int t){

return t & (t^(t-1));

}

void modify(int pos,int num)

{

while (pos<=N) {

inverse[pos]+=num;

pos+=lowbit(pos);

}

}

int sum(int end)

{

int sum=0;

while (end>0) {

sum+=inverse[end];

end-=lowbit(end);

}

return sum;

}

int main()

{

memset(inverse,0,sizeof(inverse)); //初始化

memset(count,0,sizeof(count));

char\* a="9854623870"; //长度N

for(int i=0;i<strlen(a);i++)

{

d[i+1].data =a[i]-'0';

d[i+1].pos=i+1;

}

qsort(d+1,N,sizeof(Node),cmp);

int id=1;

count[d[1].pos]=1;

for(int i=2;i<=N;i++)

{

if(d[i].data==d[i-1].data)

count[d[i].pos]=id;

else

count[d[i].pos]=++id;

}

int num=0;

for(int i=1;i<=N;i++)

{

modify(count[i],1);

num+=i-sum(count[i]);

}

cout<<num<<endl;

return 0;

}

#include <iostream>

using namespace std;

#define N 10

struct Node{

int data;

int pos;

};

Node d[N+1];

int inverse[N+1];

int count[N];

int cmp(const void\*a,const void\*b)

{

Node \*pa=(Node\*)a;

Node \*pb=(Node\*)b;

return pa->data-pb->data;

}

int lowbit(int t){

return t & (t^(t-1));

}

void modify(int pos,int num)

{

while (pos<=N) {

inverse[pos]+=num;

pos+=lowbit(pos);

}

}

int sum(int end)

{

int sum=0;

while (end>0) {

sum+=inverse[end];

end-=lowbit(end);

}

return sum;

}

int main()

{

memset(inverse,0,sizeof(inverse)); //初始化

memset(count,0,sizeof(count));

char\* a="9854623870"; //长度N

for(int i=0;i<strlen(a);i++)

{

d[i+1].data =a[i]-'0';

d[i+1].pos=i+1;

}

qsort(d+1,N,sizeof(Node),cmp);

int id=1;

count[d[1].pos]=1;

for(int i=2;i<=N;i++)

{

if(d[i].data==d[i-1].data)

count[d[i].pos]=id;

else

count[d[i].pos]=++id;

}

int num=0;

for(int i=1;i<=N;i++)

{

modify(count[i],1);

num+=i-sum(count[i]);

}

cout<<num<<endl;

return 0;

}

中间用到了排序，需要统计位于下标i处比i小的数，然后在树状数组中计算每个位置的和。

排序复杂度O(nlogn), 计算逆序数和的时候也是O(nlogn).

这里处理的是一个数的不同位，当然可以扩展到很多数。

另外，还可以用线段树的变形点树来解决，比树状数组好的是，空间节省了部分，不用排序。也是O(nlogn).

下面代码是从百度百科上找到的然后自己修改了部分测试。

view plaincopy to clipboardprint?

#include <stdio.h>

#include <string.h>

template < int N > // 表示可用区间为[0,N)，其中N必须是2的幂数;

class PointTree {

int a[ 2 \* N];

int size;

public:

void clear() { memset( this , 0 , sizeof ( \* this ));}

void add( int n)

{

int i = N + n; ++ size;

for (;i > 1 ; i /= 2 )

if ( ~ i & 1 ) a[i/ 2] ++ ;//偶数

}

int cntLs( int n) { // 统计小于

int i = N + n,c = 0 ; // 若统计小于等于则c=a;

for (; i > 1 ; i /= 2 )

if (i & 1 ) c += a[i / 2 ];//奇数

return c;

}

int cntGt( int n) { return size - a[N + n] - cntLs(n); }

void del(int n){

if(!a[n+=N])return;

--size;

for(--a[n]; n>1; n/=2)

if(~n&1)--a[n/2];

}

};

//附测试

PointTree<256> t;

int main(){

char\* p="13654287";

int x=0;

for(int i=0;i<strlen(p);i++)

{

x+=t.cntGt(p[i]-'0');

t.add(p[i]-'0');

}

printf("%d\n",x);

return 0;

}

#include <stdio.h>

#include <string.h>

template < int N > // 表示可用区间为[0,N)，其中N必须是2的幂数;

class PointTree {

int a[ 2 \* N];

int size;

public:

void clear() { memset( this , 0 , sizeof ( \* this ));}

void add( int n)

{

int i = N + n; ++ size;

for (;i > 1 ; i /= 2 )

if ( ~ i & 1 ) a[i/ 2] ++ ;//偶数

}

int cntLs( int n) { // 统计小于

int i = N + n,c = 0 ; // 若统计小于等于则c=a;

for (; i > 1 ; i /= 2 )

if (i & 1 ) c += a[i / 2 ];//奇数

return c;

}

int cntGt( int n) { return size - a[N + n] - cntLs(n); }

void del(int n){

if(!a[n+=N])return;

--size;

for(--a[n]; n>1; n/=2)

if(~n&1)--a[n/2];

}

};

//附测试

PointTree<256> t;

int main(){

char\* p="13654287";

int x=0;

for(int i=0;i<strlen(p);i++)

{

x+=t.cntGt(p[i]-'0');

t.add(p[i]-'0');

}

printf("%d\n",x);

return 0;

}

参考链接：部分来自ACM

http://blog.163.com/justly@yeah/blog/static/121037000200981473623901/