**[用树状数组解决区间查询问题](http://roba.rushcj.com/?p=510" \o "用树状数组解决区间查询问题的永久链接)**

2010年08月30日 180 views

本文扩写自郭神的《[树状数组新应用](http://roba.rushcj.com/wp-content/uploads/2010/08/A-new-application-of-BIT.pdf)》，在此表示膜拜。

树状数组的学名貌似叫做Binary Index Tree，关于它的基本应用可参考Topcoder上的这篇[Tutorial](http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=binaryIndexedTrees).

树状数组可以看作一个受限制的线段树，它维护一个数组，最经典的树状数组支持的基本操作有两个：(1)改变某一个元素的值 (2)查询某一个区间内所有元素的和。在此基础上，经过简单的变形可以变成支持另一组操作：(1)把一个区间内所有元素都加上一个值 (2)查询某一个元素的值。这两个都是已经泛滥了的东西了，在此不赘述。

简单的树状数组模型是不支持这样一组操作的：(1)把某一个区间内所有元素都加上一个值 (2)查询某一个区间内所有元素的和。当然，这个东西可以用线段树完成，但是线段树占内存比较大，写起来也比较繁（对我这种不会数据结构的人而言）。下面我们用一个改进版的树状数组完成这个任务。

首先一个观察是区间操作总可以变成从最左端开始，比如把区间[3..6]都加10，可以变成[1..6]加10, [1..2]减10。查询也类似。于是下面只关心从最左端开始的情况。定义Insert(p, d)表示把区间[1..p]都加d，Query(p)表示查询区间[1..p]之和。

我们考虑调用一次Insert(p, d)对以后的某次查询Query(q)的影响：

(1) 如果p<=q，总的结果会加上p\*d (2) 如果p>q，总的结果会加上q\*d

也就是说，Query(q)的结果来源可分为两部分，一部分是Insert(p1,d) (p1<=q)，一部分是Insert(p2,d) (p2 > q)。我们用两个数组B[], C[]分别维护这两部分信息，B[i]表示区间右端点恰好是i的所有区间的影响之和，C[i]表示区间右端点大于i的所有区间的影响之和。每当遇到 Insert时，考虑当前的Insert会对以后的Query产生什么影响，更新B和C数组；当遇到Query时，把两部分的结果累加起来。

具体来说，当我们遇到Insert(p, d)时，把B[p]增加p\*d，把C[1], C[2], …, C[p-1]都增加d。当遇到Query(p)时，查询B[1]+B[2]+…+B[p]+C[p]\*p即可。可以发现对B数组是修改单个元素，查询区间和；对C数组是修改区间，查询单个元素，这恰好对应于一开始说的树状数组支持的基本操作。于是我们用两个树状数组漂亮地完成了任务。:)

示例代码：

#include <cstdio>

const int MAXN = 1024;

int B[MAXN], C[MAXN];

#define LOWBIT(x) ((x)&(-(x)))

void bit\_update(int \*a, int p, int d) {

for ( ; p && p < MAXN ; p += LOWBIT(p))

a[p] += d;

}

int bit\_query(int \*a, int p) {

int s = 0;

for ( ; p ; p -= LOWBIT(p))

s += a[p];

return s;

}

void bit\_update2(int \*a, int p, int d) {

for ( ; p ; p -= LOWBIT(p))

a[p] += d;

}

int bit\_query2(int \*a, int p) {

int s = 0;

for ( ; p && p < MAXN ; p += LOWBIT(p))

s += a[p];

return s;

}

inline void \_insert(int p, int d) {

bit\_update(B, p, p\*d);

bit\_update2(C, p-1, d);

}

inline int \_query(int p) {

return bit\_query(B, p) + bit\_query2(C, p) \* p;

}

inline void insert\_seg(int a, int b, int d) {

\_insert(a-1, -d);

\_insert(b, d);

}

inline int query\_seg(int a, int b) {

return \_query(b) - \_query(a-1);

}

int main() {

int com, a, b, c;

while (scanf("%d%d%d",&com,&a,&b) != EOF) {

a += 2; b += 2; //防止出现负数

if (com == 0) { //更新

scanf("%d",&c);

insert\_seg(a, b, c);

} else { //查询

printf("%d**\n**",query\_seg(a,b));

}

}

return 0;

}