[BZOJ 3110] [Zjoi2013] K大数查询 【树套树】

**Description**

有N个位置，M个操作。操作有两种，每次操作如果是1 a b c的形式表示在第a个位置到第b个位置，每个位置加入一个数c  
如果是2 a b c形式，表示询问从第a个位置到第b个位置，第c大的数是多少。

**Input**

第一行N，M  
接下来M行，每行形如1 a b c或2 a b c

**Output**

输出每个询问的结果

**Sample Input**

2 5  
1 1 2 1  
1 1 2 2  
2 1 1 2  
2 1 1 1  
2 1 2 3

**Sample Output**

1  
2  
1

**HINT**

N,M<=50000,N,M<=50000a<=b<=N  
1操作中abs(c)<=N  
2操作中abs(c)<=Maxlongint

第一想法是外层线段树内层treap名次树。。嗯这样写的话大概就是个暴力  = = TLE不说。。内存估计都要爆（即使有512M内存）

后来发现这玩意其实线段树套线段树就可以解决 于是我就orz了 = =

外层不是区间。外层写权值线段树（就是每次add进去的值） 内层套区间（add进去的值在哪些地方） 然后就可以放lazy标记了 每次更新的时候lazy++

当然好像普通的lazy标记只能卡着过。。所以我们可以放永久化标记……（其实直接转zkw最好……就是那玩意太高端了弱比表示不会……）

永久化标记就是lazy不pushdown 直接在询问的时候把lazy加在答案里（……语死早）

对于每个询问 先查找右边的总数cnt 如果右边总数大于等于c就在右子树查询第c大的 如果不够就在左子树查询第c-cnt大的。。

内存的问题呢。。外层线段树开个4\*maxn的root数组就可以了。。然后内层根本不用建出来 每次动态分配1下就好了（当然还是先把内存申请好 = =动态申请太慢了）

内层线段树的空间我算出来是17600000这样。。就是4 \* (logn)^2 \* 4 \*n = =反正512M嘛应该都是够用的。。第一次线段树没写struct。。反正写法其实也差不多。。。然后就是内外层线段树一定要分清。。（逼得我开始打注释了orz）

无聊写了个读入优化 orz其实好像也没有快多少的样子。。bzoj交上去差了20+ms。。。。。 = =

然后我cena的迷之WA是因为query又忘了返回了 = =dev神奇的补return功能简直无法恭维。。。还有bzoj也是…………

#include <cstdio>

#include <iostream>

#define lch (u << 1)

#define rch (u << 1 | 1)

using namespace std;

int ReadInt(){

int x = 0, sign = 1; char ch = getchar();

while(ch < '0' || ch > '9') { if(ch == '-') sign = -1; ch = getchar(); }

while(ch >= '0' && ch <= '9') {x = x\*10 + ch-'0'; ch = getchar();}

return x\*sign;

}

const int maxn = 50000+5;

const int maxm = 360 \* maxn;

int N, M;

int root[3\*maxn]; // 外层线段树

int sum[maxm], lazy[maxm], lc[maxm], rc[maxm]; // 内层线段树

int size;

void insert(int u, int l, int r, int L, int R){ // 内层线段树，值在线段中分布。

if(l == L && r == R){

sum[u] += r-l+1;

lazy[u]++; return;

}

int mid = (l + r) >> 1;

if(R <= mid) insert(lc[u] ? lc[u] : lc[u] = ++size, l, mid, L, R);

else if(L > mid) insert(rc[u] ? rc[u] : rc[u] = ++size, mid+1, r, L, R);

else {

insert(lc[u] ? lc[u] : lc[u] = ++size, l, mid, L, mid);

insert(rc[u] ? rc[u] : rc[u] = ++size, mid+1, r, mid+1, R);

}

sum[u] += R - L + 1;

}

void add(int u, int l, int r, int L, int R, int x){ // 外层线段树，值域

insert(root[u] ? root[u] : root[u] = ++size, 1, N, L, R);

if(l == r) return;

int mid = (l + r) >> 1;

if(x <= mid) add(lch, l, mid, L, R, x);

else add(rch, mid+1, r, L, R, x);

}

int count(int u, int l, int r, int L, int R){ // 内层线段树

if(l == L && r == R) return sum[u];

int mid = (l + r) >> 1, ans = 0;

if(R <= mid) ans = count(lc[u], l, mid, L, R);

else if(L > mid) ans = count(rc[u], mid+1, r, L, R);

else{ ans += count(lc[u], l, mid, L, mid);

ans += count(rc[u], mid+1, r, mid+1, R);

}

ans += (R - L + 1) \* lazy[u];

return ans;

}

int query(int u, int l, int r, int L, int R, int x){ // 外层线段树

if(l == r) return l;

int mid = (l + r) >> 1;

int cnt = count(root[rch], 1, N, L, R);

if(cnt >= x) return query(rch, mid+1, r, L, R, x);

else return query(lch, l, mid, L, R, x - cnt);

}

int main(){

N = ReadInt(); M = ReadInt();

int type, a, b, c;

while(M--){

type = ReadInt(); a = ReadInt(); b = ReadInt(); c = ReadInt();

if(type == 1) add(1, 1, N, a, b, c);

else printf("%d\n", query(1, 1, N, a, b, c));

}

return 0;

}

第一层线段树是权值线段树，它的每个节点对应了整个区间的线段树，并且第二层的线段树维护了第一层权值区间中所有数出现的总次数。

插入一种颜色c的时候，在第一层相当于单点修改，会递归地影响线段树上O(logn)个结点，对于影响到的每个节点，都只需要再在第二层线段树上做一次区间修改即可（之所以要将权值放在第一层，就是因为要在第二层线段树上的节点打lazy标记，而我实在是想不出来第一层的线段树该如何处理lazy的下放问题……）。为了减少常数，不妨在实现时使lazy标记永久化。

查询时，用类似于二分查找的办法（这个貌似跟主席树是类似的），假设当前已经确定了答案的区间为[l,r]，设其中点为m，第二层线段树查询的区间为[ql,qr]，由于是查询第k大的数，我们便查询在[ql,qr]中，所有在[m+1,r]范围内的颜色出现的总次数t，若t>=k，则说明答案在右区间；反之，说明答案在左区间，同时将k减去t。

代码真是比想象中的短多了……

#include<iostream>

#include<cstdio>

#define maxn 50000

#define maxm 20000000

#define mid (l+r>>1)

using namespace std;

int lc[maxm+10],rc[maxm+10],sum[maxm+10],val[maxm+10],root[(maxn<<4)+10],ql,qr,cnt;

void update(int &o,int l,int r){

if(!o)o=++cnt;

if(ql<=l&&r<=qr)val[o]++,sum[o]+=r-l+1;

else{

int m=mid;

if(ql<=m)update(lc[o],l,m);

if(qr>m)update(rc[o],m+1,r);

sum[o]=sum[lc[o]]+sum[rc[o]]+val[o]\*(r-l+1);

}

}

void query(int &o,int l,int r,int &qsum,int add){

if(!o)o=++cnt;

if(ql<=l&&r<=qr)qsum+=sum[o]+add\*(r-l+1);

else{

int m=mid;

if(ql<=m)query(lc[o],l,m,qsum,add+val[o]);

if(qr>m)query(rc[o],m+1,r,qsum,add+val[o]);

}

}

int main(){

int n,q;

scanf("%d%d",&n,&q);

int k,a,b,c;

while(q--){

scanf("%d%d%d%d",&k,&ql,&qr,&c);

int o=1,l=1,r=n;

if(k==1){

for(;;){

update(root[o],1,n);

if(l==r)break;

int m=mid;

if(c<=m)o=o<<1,r=m;

else o=o<<1|1,l=m+1;

}

}else{

int t;

for(;;){

if(l==r)break;

int m=mid;

query(root[o<<1|1],1,n,t=0,0);

if(t>=c){

o=o<<1|1;

l=m+1;

}else{

c-=t;

o=o<<1;

r=m;

}

}

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

}

}

return 0;

}



