**线段树小结 收藏**

这段时间是线段树专题。做了好些题目，发现线段树实在是太灵活了，可以根据题目所需添加一些信息或者调整一些构造或查询方法。 小结一下。

线段树用途：

RMQ，线段求长，矩形交，矩形并等……

线段树基本操作：

建树，插入，删除，查询，更新，删树

因为它是一棵二叉树，所以它的操作一般除了建树，删树是O（N），其余的都是O（LogN）的。

这个复杂度基本能顺利解决卡时的问题。

建树和删树：

这两个是相互影响的过程，有些删树只需要O（1）就行了。

1）节点的内存分配问题

a. 使用动态内存分配，new或者malloc，如果这样，删树往往得消耗O（N）的时间都delete或free

b. 计算节点个数上限，开一个静态的大数组，并且加一个下标或指针来模拟分配。删树时只要将指针或下标回复下即可，单位时间内就实现删树过程。

node mem[MAX];

int mem\_pos = 0;

node \*

new\_node()

...{

node \* pt = &mem[mem\_pos ++];

memset(pt,0,sizeof(node));

return pt;

}

2）节点信息问题

一般线段树处理区间问题，所以它的信息必定包含左右端点的值，左右子树的指针，如果是RMQ可以加个状态值，如果是处理线段的话，那加的比较多了，一般是覆盖数，测度，重复测度，独立线段数，权和……

总之可以依据题目所需，灵活添加。

但是得注意的是，左右端点值必须是整数，因为实数一是存在精度问题，最重要的是分区间的问题，实数除不尽啊。但是也有解决方法，如果区间数有限的话，可以将实数存在数组里面，用数组下标来表示，这样就能实现实数和整数的映射。而查找也得付出时间消耗，二分也得O（LogN）。

3）树的表示问题

对于有些静态RMQ，它给予了初始的状态，则可以根据所有状态一次性建造线段树

如：http://acm.hdu.edu.cn/showproblem.php?pid=1754 I Hate It

而像有些未给定初始状态，只是每次会添加一些信息的，可以开始时只建空树。

如：http://acm.hdu.edu.cn/showproblem.php?pid=1823 Luck and Love

上面的题目虽然节点会添加，但是也可以先预先建造一颗逻辑上的线段树结构，保存的信息为空以便之后更新，这需要你知道他的范围以后才可以建造。把这种想法应用到删树的时候，也可以只清空信息域，保留树的逻辑，这样的过程也相当于删除了树，并且省去了下次建树的时间消耗。

插入，删除，更新，查询：

简单的更新到点就不用多说了，依据子树的划分确定是访问哪个子树，递归下去，修改叶节点信息，回溯时修改父节点的信息。写起来也是很快的。

说下成段更新，就是更新的也是一个区间，而且该区间较大，更新到点明显就是太耗时了，时间复杂度是O（N）的。

看下http://acm.hdu.edu.cn/showproblem.php?pid=1779 这是内部题目，描述见尾部。

成段更新，并且还得找出最大值的那位，将他的值改为0。

我的做法是，定义三个信息域，mmax保存他子区间的最大值，imax子区间最大值的位置，add本区间累加数。

当找到能完全覆盖它的区间后，在add上加上所给的蜡烛数，回溯时，在他的父节点判断左右子树的值，记录最大值和左右子树中最大值的位置。在父节点判断用的值是子树的add+mmax值。这样就避免了更新到点，只是在经过的路径上累加被加的值。这样就可以保证他的更新复杂度任为O（LogN）。

update(更新区间[l,r], 更新值v)

...{

if(当本区间刚好是更新区间[l,r]) ...{

add += v;

return;

}

mid = (l+r)/2;

update(左子区间[l,mid], 更新值v);

update(右子区间[mid+1,r], 更新值v);

if(左子区间mmax+add >= 右子区间mmax+add) ...{

mmax = 左子区间mmax+add;

imax = 左子区间imax;

}

else ...{

mmax = 右子区间mmax+add;

imax = 右子区间imax;

}

return;

}

线段树一般都是用来做离散化的，再来看看经典的题目吧。

http://acm.hdu.edu.cn/showproblem.php?pid=1542 Atlantis（矩形交面积）

http://acm.hdu.edu.cn/showproblem.php?pid=1255 覆盖的面积（矩形并面积）

http://acm.hdu.edu.cn/showproblem.php?pid=1828 Picture（矩形轮廓周长）

涉及到二维空间，可以使用二维线段树求解。但是以上3题都可以用一维线段树，另一维经过排序离散化后，按次序扫描，即时间复杂度是O（NLogN）。说下几个上述几题要使用到的信息域。

覆盖数：区间被完全覆盖的次数

测度：区间中线段覆盖的长度

重复测度：区间中覆盖超过一次的线段长度

独立线段树：区间中互不相交的线段数

但二维静态RMQ时，如果只使用一维线段树，那只是从O（N^2）降低成O（NLogN）而已，对于大的数据量来说，还是太慢了。

如：http://acm.zju.edu.cn/show\_problem.php?pid=2859 Matrix Searching

使用二维线段树的话，查询时间复杂度可以降为O（LogN LogN），相当的有效了。其实还有O（1）的做法，因为是静态的RMQ，大小桶之类的做法完全可以更为高效。

二维线段树的方法其实也很简单，在节点中再加个 node \*类型的指针，指向第二维线段树。

由此可见灵活使用线段树能解决很多复杂问题，而且易于扩充其功能，是居家旅行，必备良药o(∩\_∩)o...哈哈

Problem C(1003)

Happy Children’s Day

Standard Input / Standard Output

Children's Day is coming. In this day, children will get a lot of candies. In MAX city, people develop an automatic candy management (ACM) system. ACM can manage N piles of candies. The system can carry out two operations.

1).I a b c (1<=a<=b<=N, 0<c<=100), ACM will add each pile numbered from a to b with c candies.

2).C a b (1<=a<=b<=N), ACM will choose the pile (from a to b) with the most candies, and give all the candies in that pile to one child. If two or more piles have the most candies, then the pile with smaller id will be chosen.

Given a series of instructions, for each operation C a b, please help the people to find out the number of candies a child can get.

Input:

Input may contain several test data sets.

For each data set, the first line contains two integers N, M(0<N,M<=10^5) where N indicates the number of piles, and M indicates the number of operations that follows.

Each of the next M lines describes one operation.

Input will be ended by N=0,M=0,which should not be processed.

NOTE: At the beginning of each data set, all of the N piles have 0 candies.

Output:

For each operation C a b, print the number of candies the child can get in a line.

Sample Input:

5 4

I 1 5 1

C 2 3

I 2 2 4

C 2 3

0 0

Sample Output:

1

4

附上几篇源码：

//Problem : 1779 Happy Children’s Day

#include <cstdio>

#include <string>

#include <algorithm>

using namespace std;

struct node ...{

int l,r;

node \* pl, \* pr;

int imax;//子区间最大值位置

int add;//本区间累加值

int mmax;//记录子区间最大值

}mem[300000];

int mem\_pos;

node \*

new\_node()

...{

node \* pt = &mem[mem\_pos ++];

memset(pt,0,sizeof(node));

return pt;

}

node \*

make\_tree(int il, int ir)

...{

node \* root = new\_node();

root ->l = il;

root ->r = ir;

root ->imax = il;

if(ir > il) ...{

int mid = (il+ir)/2;

root ->pl = make\_tree(il, mid);

root ->pr = make\_tree(mid+1, ir);

}

return root;

}

int

update(node \* root,int l, int r, int v)

...{

if(root ->l == l && root ->r == r) ...{

root ->add += v;

return root ->mmax + root ->add;

}

int mid = (root ->l + root ->r)/2;

if(l <= mid) ...{

if(r <= mid) ...{

int t = update(root ->pl, l, r, v);

if(t >= root ->pr ->add + root ->pr ->mmax) ...{

root ->mmax = t;

root ->imax = root ->pl ->imax;

}

else ...{

root ->mmax = root ->pr ->add + root ->pr ->mmax;

root ->imax = root ->pr ->imax;

}

}

else ...{

root ->mmax = update(root ->pl, l, mid, v);

root ->imax = root ->pl ->imax;

int t = update(root ->pr, mid+1, r, v);

if(t > root ->mmax) ...{

root ->mmax = t;

root ->imax = root ->pr ->imax;

}

}

}

else ...{

int t = update(root ->pr, l, r, v);

if(t > root ->pl ->add + root ->pl ->mmax) ...{

root ->mmax = t;

root ->imax = root ->pr ->imax;

}

else ...{

root ->mmax = root ->pl ->add + root ->pl ->mmax;

root ->imax = root ->pl ->imax;

}

}

return root ->mmax + root ->add;

}

int

get(node \* root,int l, int r,int &imax)

...{

if(root ->l == l && root ->r == r) ...{

imax = root ->imax;

return root ->mmax + root ->add;

}

int mid = (root ->l + root ->r)/2;

int t1,t2;

if(l <= mid) ...{

if(r <= mid) ...{

t1 = get(root ->pl, l,r,imax);

}

else ...{

int imax2;

t1 = get(root ->pl, l,mid,imax);

t2 = get(root ->pr, mid+1,r,imax2);

if(t2 > t1) ...{

t1 = t2;

imax = imax2;

}

}

}

else ...{

t1 = get(root ->pr, l,r,imax);

}

return t1 + root ->add;

}

int main()

...{

int t,n,m,i,j;

int a,b,c;

char cmd[5];

int imax;

mem\_pos = 0;

node \* root = make\_tree(1, 100010);

while(scanf("%d %d", &n,&m)==2) ...{

if(n==0 && m==0) ...{

break;

}

for(i=0;i<mem\_pos;i++) ...{

mem[i].imax = mem[i].l;

mem[i].add = mem[i].mmax = 0;

}

getchar();

while(m --) ...{

scanf("%s",cmd);

if(cmd[0] == 'I') ...{

scanf("%d %d %d",&a,&b,&c);

update(root, a,b,c);

}

else ...{

int cand;

scanf("%d %d",&a,&b);

imax = 0;

printf("%d ",cand = get(root, a,b,imax));

if(imax != 0) ...{

update(root, imax, imax, -cand);

}

//printf("get pile = %d, num = %d ", imax,cand);

}

}

}

}//Problem : 1828 ( Picture )

#include <cstdio>

#include <string>

#include <algorithm>

using namespace std;

struct node ...{

int l, r;//左右端点

int il, ir;

node \* pl, \* pr;//左右子树

int len;//测度

int cover;//完全覆盖线段数

int lines;//独立线段数

int lbd, rbd;//左右端点覆盖

}mem[40000];

int mem\_pos;

struct line ...{

int x,y1,y2;

int flag;

bool operator < (const line & tt ) const ...{

if(x != tt.x) ...{

return x < tt.x;

}

return flag > tt.flag;

}

}list[11000];

int yaxis[11000];

node \*

new\_node()

...{

node \* pt = &mem[mem\_pos ++];

memset(pt,0,sizeof(node));

return pt;

}

void

cal\_len(node \* root)

...{

root ->len = 0;

if(root ->cover > 0) ...{

root ->len = root ->r - root ->l;

}

else ...{

if(root ->pl && root ->pr) ...{

root ->len = root ->pl ->len + root ->pr ->len;

}

}

}

void

cal\_line(node \* root)

...{

root ->lbd = root ->rbd = root ->lines = 0;

if(root ->cover > 0) ...{

root ->lbd = root ->rbd = root ->lines = 1;

}

else ...{

if(root ->pl && root ->pr) ...{

root ->lbd = root ->pl ->lbd;

root ->rbd = root ->pr ->rbd;

root ->lines = root ->pl ->lines + root ->pr ->lines

- (root ->pl ->rbd \* root ->pr ->lbd);

}

}

}

node \*

make\_tree(int axis[], int il, int ir)

...{

node \* root = new\_node();

root ->l = axis[il];

root ->r = axis[ir];

root ->il = il;

root ->ir = ir;

if(ir - il > 1) ...{

int mid = (il+ir)/2;

root ->pl = make\_tree(axis, il, mid);

root ->pr = make\_tree(axis, mid, ir);

}

return root;

}

void

update(node \* root, line e)

...{

if(e.y1 <= root ->l && root ->r <= e.y2) ...{

root ->cover += e.flag;

}

if(root ->ir - root ->il > 1) ...{

int mid = (root ->il + root ->ir)/2;

if(e.y1 < yaxis[mid]) ...{

update(root ->pl, e);

}

if(yaxis[mid] < e.y2) ...{

update(root ->pr, e);

}

}

cal\_len(root);

cal\_line(root);

}

int main()

...{

int t,n,i;

int ls;

int x1,x2,y1,y2;

int sum;

while(scanf("%d", &n)==1) ...{

ls = 0;

mem\_pos = 0;

sum = 0;

for(i=0;i<n;i++) ...{

scanf("%d %d %d %d", &x1,&y1,&x2,&y2);

list[ls].x = x1;

list[ls].y1 = y1; list[ls].y2 = y2;

list[ls].flag = 1;

list[ls+1].x = x2;

list[ls+1].y1 = y1; list[ls+1].y2 = y2;

list[ls+1].flag = -1;

yaxis[ls] = y1;

yaxis[ls+1] = y2;

ls += 2;

}

sort(yaxis, yaxis+ls);

sort(list, list+ls);

int preLen = 0, preLines = 0;

node \* root = make\_tree(yaxis, 0, ls-1);

for(i=0;i<ls;i++) ...{

update(root, list[i]);

sum += abs(root ->len - preLen);

sum += 2 \* preLines \* (list[i].x - list[i-1].x);

//printf("(%d,%d) %d (%d,%d) ", root->len, root->lines, sum, preLen,preLines);

preLen = root ->len;

preLines = root ->lines;

}

printf("%d ", sum);

}

}//Problem : 1255 ( 覆盖的面积 )

#include <cstdio>

#include <string>

#include <algorithm>

using namespace std;

struct node ...{

double l,r;

node \* pl, \* pr;

double len;//测度

int cover;//完全覆盖数

double relen;//重复测度

}mem[10000];

int mem\_pos;

struct line ...{

double x,y1,y2;

int flag;

bool operator < (const line & tt ) const ...{

return x < tt.x;

}

}list[2100];

double yaxis[2100];

node \*

new\_node()

...{

node \* pt = &mem[mem\_pos ++];

memset(pt,0,sizeof(node));

return pt;

}

void

cal\_len(node \* root)

...{

root ->len = 0;

if(root ->cover > 0) ...{

root ->len = root ->r - root ->l;

}

else ...{

if(root ->pl) ...{

root ->len += root ->pl ->len;

}

if(root ->pr) ...{

root ->len += root ->pr ->len;

}

}

}

void

cal\_relen(node \* root)

...{

root ->relen = 0;

if(root ->cover > 1) ...{//当本线段被完全覆盖多次

root ->relen = root ->r - root ->l;

}

else if(root ->cover == 1) ...{//当本线段被完全覆盖一次

if(root ->pl) ...{

root ->relen += root ->pl ->len;

}

if(root ->pr) ...{

root ->relen += root ->pr ->len;

}

}

else ...{//未被覆盖过

if(root ->pl) ...{

root ->relen += root ->pl ->relen;

}

if(root ->pr) ...{

root ->relen += root ->pr ->relen;

}

}

}

node \*

make\_tree(int il, int ir)

...{

node \* root = new\_node();

root ->l = yaxis[il];

root ->r = yaxis[ir];

if(ir - il > 1) ...{

int mid = (il+ir)/2;

root ->pl = make\_tree(il, mid);

root ->pr = make\_tree(mid, ir);

}

return root;

}

void

update(node \* root, line e)

...{

if(root ->l == e.y1 && root ->r == e.y2) ...{

root ->cover += e.flag;

cal\_len(root);

cal\_relen(root);

return ;

}

if(root ->pl ->r > e.y1) ...{//[l,r)

if(root ->pl ->r >= e.y2) ...{

update(root ->pl, e);

}

else ...{

line e2 = e;

e2.y2 = root ->pl ->r;

update(root ->pl, e2);

e2 = e;

e2.y1 = root ->pr ->l;

update(root ->pr, e2);

}

}

else ...{

update(root ->pr, e);

}

cal\_len(root);

cal\_relen(root);

}

int main()

...{

int t,n,i;

int ls;

double x1,x2,y1,y2;

double sum, unions, intersections;

scanf("%d", &t);

while(t --) ...{

ls = 0;

mem\_pos = 0;

sum = unions = intersections = 0;

scanf("%d", &n);

for(i=0;i<n;i++) ...{

scanf("%lf %lf %lf %lf", &x1,&y1,&x2,&y2);

list[ls].x = x1;

list[ls].y1 = y1; list[ls].y2 = y2;

list[ls].flag = 1;

list[ls+1].x = x2;

list[ls+1].y1 = y1; list[ls+1].y2 = y2;

list[ls+1].flag = -1;

yaxis[ls] = y1;

yaxis[ls+1] = y2;

ls += 2;

sum += (x2 - x1)\*(y2 - y1);

}

sort(yaxis, yaxis+ls);

sort(list, list+ls);

node \* root = make\_tree(0, ls-1);

update(root, list[0]);

for(i=1;i<ls;i++) ...{

intersections += root ->relen \* (list[i].x - list[i-1].x);

update(root, list[i]);

}

printf("%.2lf ", intersections);

}

}//Problem : 1542 ( Atlantis )

#include <cstdio>

#include <string>

#include <algorithm>

using namespace std;

struct node ...{

double l,r;

node \* pl, \* pr;

double len;

int cover;

}mem[10000];

int mem\_pos;

struct line ...{

double x,y1,y2;

int flag;

bool operator < (const line & tt ) const ...{

return x < tt.x;

}

}list[2100];

double yaxis[2100];

node \*

new\_node()

...{

node \* pt = &mem[mem\_pos ++];

memset(pt,0,sizeof(node));

return pt;

}

void

cal\_len(node \* root)

...{

root ->len = 0;

if(root ->cover > 0) ...{

root ->len = root ->r - root ->l;

}

else ...{

if(root ->pl) ...{

root ->len += root ->pl ->len;

}

if(root ->pr) ...{

root ->len += root ->pr ->len;

}

}

}

node \*

make\_tree(int il, int ir)

...{

node \* root = new\_node();

root ->l = yaxis[il];

root ->r = yaxis[ir];

if(ir - il > 1) ...{

int mid = (il+ir)/2;

root ->pl = make\_tree(il, mid);

root ->pr = make\_tree(mid, ir);

}

return root;

}

void

update(node \* root, line e)

...{

if(root ->l == e.y1 && root ->r == e.y2) ...{

root ->cover += e.flag;

cal\_len(root);

return ;

}

if(root ->pl ->r > e.y1) ...{//[l,r)

if(root ->pl ->r >= e.y2) ...{

update(root ->pl, e);

}

else ...{

line e2 = e;

e2.y2 = root ->pl ->r;

update(root ->pl, e2);

e2 = e;

e2.y1 = root ->pr ->l;

update(root ->pr, e2);

}

}

else ...{

update(root ->pr, e);

}

cal\_len(root);

}

int main()

...{

int t,n,i;

int ls;

double x1,x2,y1,y2;

double sum, unions, intersections;

//scanf("%d", &t);

t = 1;

while(1) ...{

ls = 0;

mem\_pos = 0;

sum = unions = intersections = 0;

scanf("%d", &n);

if(n==0) break;

for(i=0;i<n;i++) ...{

scanf("%lf %lf %lf %lf", &x1,&y1,&x2,&y2);

list[ls].x = x1;

list[ls].y1 = y1; list[ls].y2 = y2;

list[ls].flag = 1;

list[ls+1].x = x2;

list[ls+1].y1 = y1; list[ls+1].y2 = y2;

list[ls+1].flag = -1;

yaxis[ls] = y1;

yaxis[ls+1] = y2;

ls += 2;

sum += (x2 - x1)\*(y2 - y1);

}

sort(yaxis, yaxis+ls);

sort(list, list+ls);

node \* root = make\_tree(0, ls-1);

update(root, list[0]);

for(i=1;i<ls;i++) ...{

unions += root ->len \* (list[i].x - list[i-1].x);

update(root, list[i]);

}

printf("Test case #%d Total explored area: %.2lf ", t ++, unions);

//printf("%.2lf ", sum - unions);

}

}//Matrix Searching

#include <cstdio>

#include <string>

#include <algorithm>

using namespace std;

int t,n,q;

int matrix[310][310];

struct node ...{

int l,r;

node \* pl, \* pr;

node \* ytree;

int mmin;

}mem[400000];

int mem\_pos;

node \*

new\_node()

...{

node \* pt = &mem[mem\_pos ++];

memset(pt,0,sizeof(pt));

return pt;

}

node \*

make\_tree(int x1,int x2,int y1,int y2,bool flag)

...{

node \* root = new\_node();

if(flag) ...{// first dimension

root ->l = x1;

root ->r = x2;

root ->ytree = make\_tree(x1,x2,y1,y2,false);

if(x1 != x2) ...{

int mid = (root ->l + root ->r)/2;

root ->pl = make\_tree(x1, mid,y1,y2,true);

root ->pr = make\_tree(mid+1, x2,y1,y2,true);

}

}

else ...{// second dimension

root ->l = y1;

root ->r = y2;

if(y1 != y2) ...{

int mid = (root ->l + root ->r)/2;

root ->pl = make\_tree(x1,x2,y1,mid,false);

root ->pr = make\_tree(x1,x2,mid+1,y2,false);

root ->mmin = min(root ->pl ->mmin, root ->pr ->mmin);

}

else ...{

root ->mmin = matrix[x1][y1];

int i;

for(i=x1+1;i<=x2;i++) ...{

root ->mmin = min(root ->mmin, matrix[i][y1]);

}

}

}

return root;

}

int

query(node \* root, int x1,int x2,int y1,int y2,bool flag)

...{

int t1,t2;

int mid = (root ->l + root ->r)/2;

if(flag) ...{// first dimension

if(root ->l == x1 && root ->r == x2) ...{

return query(root ->ytree, x1,x2,y1,y2,false);

}

else ...{

if(x2 <= mid) ...{

return query(root ->pl,x1,x2,y1,y2,true);

}

if(x1 > mid) ...{

return query(root ->pr,x1,x2,y1,y2,true);

}

t1 = query(root ->pl,x1,mid,y1,y2,true);

t2 = query(root ->pr,mid+1,x2,y1,y2,true);

}

}

else ...{// second dimension

if(root ->l == y1 && root ->r == y2) ...{

return root ->mmin;

}

else ...{

if(y2 <= mid) ...{

return query(root ->pl,x1,x2,y1,y2,false);

}

if(y1 > mid) ...{

return query(root ->pr,x1,x2,y1,y2,false);

}

t1 = query(root ->pl,x1,x2,y1,mid,false);

t2 = query(root ->pr,x1,x2,mid+1,y2,false);

}

}

return min(t1,t2);

}

int main()

...{

int i,j,k;

int c1,c2,r1,r2;

scanf("%d",&t);

while(t --) ...{

scanf("%d", &n);

for(i=1;i<=n;i++) ...{

for(j=1;j<=n;j++) ...{

scanf("%d", &matrix[i][j]);

}

}

mem\_pos = 0;

node \* root = make\_tree(1,n,1,n,true);

scanf("%d", &q);

while(q --) ...{

scanf("%d %d %d %d",&r1,&c1,&r2,&c2);

printf("%d ", query(root,r1,r2,c1,c2,true));

}

}

}