**伸展树（转载http://acm.tju.edu.cn/blog/?p=85） 收藏**

前段日子写过 treap 与 左偏树， 这两天又看了伸展树小有心得就在此记下来以备来日参考

左偏树是一个斜堆，而treap 是 tree 与 heap 的混合产物即是一个堆又一个二叉查找树，代码量很小易调试很合适上场比赛。但有些问题比如较复杂的操作上面的这些数据结构搞不定的话我们可以换 splay 来试试。

splay作为一个二叉排序树所具有特点用一句来概述就是“可以让你把指的结点放在树根的位置或者作为树根的儿子”很多参考资料都会这么介绍 splay 说它是一个自调整树，把最近访问的节点移到根处以使得下次再访问它时可以快速找到，我理解的 splay 就只有一种操作就是把一个节点移动到根处，在这个过程中我们不需要处理平衡的问题，尽管它可能会变成一个线性的形状但这都是小概率事件

就我目前的知识而言 splay 强大之处在于对它可以快速的对区间进行操作，比如对一个序列的指定子序列进行反转或者把它们整体移动到一个新的位置（参考TOJ-3578）.。还对在区间的指定位置处随意插入、删除、更新元素，同时可以很快取得指定子区间的和、指定子区间的连续或者区间内的最大、最小值（参考SPOJ-4487）

初看起来询问区间最大最小值与子区间和或者连续最大和 这些不都是线段树干的活。是的你没有想错，在学会 splay 之前，线段树的确是处理这个区间问题的高手，但在你完全掌握 splay 后，它能让你处理更多的区间操作，对一些问题它内功要远远高于线段树，至于一些复杂问题都到了非用它不可的地步。

说了这么多废话现在来说说它的魅力之处吧：假设你已经知道二叉查找树的左右旋转操作，那么你一定可以

把树中指定的一个节点旋转到根部，这就是 splay 的看家本领，别的扩展功能全依靠它了。先说下如何在一个指定的位置插入一个值，假设有数列 1，2，3，4，5，6，7为元素的 splay树，如果我们要在第3个位置 与第4个位置（即3与4）之间插入一个值为 val 的新元素 应该进行的操作是：

（1）把元素3旋转到根处。

（2）把元素4旋转到根节点下面即作为3的儿子。

这时4一定是3的右儿子，并且它的左子是空的，现在就新建一个以 val 为值的新元素挂在根的右子的左子上面（不明白的动手画下图）。现在如果对 splay 树进行先序遍历的话是不是发现 val 已经在 3 与4之间了。

同样的道理可以把一个新区间插入一个指定的位置上或者把一个指定的值与指的区间删除，方法都是进行两个旋转然后要被处理的元素或者区间就神奇地程现在根的右子的左子那里，如果还有其它的统计信息可以记录在相应的区间的根处，当 splay 操作对它进行旋转时注意实时更新就可以了

有兴趣的就赶快把这两道题写了吧

https://www.spoj.pl/problems/GSS6/

http://acm.tju.edu.cn/toj/showp3578.html

下面是 spoj 4487 的代码，可以用来做模板用

#include <iostream>

#include <cstring>

#include <algorithm>

#include <vector>

#include <cmath>

#include <string>

#include <map>

#include <fstream>

using namespace std;

typedef long long ll;

const int MAXN = 102000 \* 2;

const int INF = (1 << 28);

int ary[MAXN];

struct Node{

int val, sz;

int sum, ml, mr, mc;

Node \*ch[2], \*pre;

};

class Splay{

public:

Node \*null, \*root, \*S[MAXN], data[MAXN];

int cnt, top;

Node \* getNewNode(int x){

Node \*p;

if(top){

p = S[top--];

}else{

p = &data[cnt++];

}

p->val = p->sum = p->ml = p->mr = p->mc = x;

p->sz = 1;

p->ch[0] = p->ch[1] = p->pre = null;

return p;

}

void init(){

cnt = top = 0;

null = getNewNode(-INF);

null->sum = null->sz = 0;

root = getNewNode(-INF);

root->sum = 0;

root->ch[1] = getNewNode(INF);

root->ch[1]->sum = 0;

root->ch[1]->pre = root;

updata(root);

}

void updata(Node \*p){

p->sz = p->ch[0]->sz + p->ch[1]->sz + 1;

p->sum = p->ch[0]->sum + p->ch[1]->sum + p->val;

//这玩意怎么和线段树那么像啊！

p->ml = max(p->ch[0]->ml, p->ch[0]->sum + p->val + max(p->ch[1]->ml, 0));

p->mr = max(p->ch[1]->mr, p->ch[1]->sum + p->val + max(p->ch[0]->mr, 0));

//ml 左边连续的最大值, mr 右边连续的最大值, mc 该点作为根时连续的最大值

p->mc = max(p->ch[0]->mc, p->ch[1]->mc);

p->mc = max(p->mc, max(p->ch[0]->mr + p->ch[1]->ml, 0) + p->val);

p->mc = max(p->mc, max(p->ch[0]->mr, p->ch[1]->ml) + p->val);

}

void rotate(Node \*x, int c){

//c = 1 是右旋 0 是左旋 , 将 x 节点旋转到它父亲的位置

Node \*y = x->pre;

y->ch[!c] = x->ch[c];

if(x->ch[c] != null){

x->ch[c]->pre = y;

}

x->pre = y->pre;

if(y->pre != null){

if(y->pre->ch[0] == y){

y->pre->ch[0] = x;

}else {

y->pre->ch[1] = x;

}

}

x->ch[c] = y;

y->pre = x;

if(y == root){

root = x;

}

updata(y);

}

void splay(Node \*x, Node \*f){

//把节点 x 旋转为 f 的儿子，若 f 为 null 则把 x 旋转到根的位置

while(x->pre != f){

if(x->pre->pre == f){//zig

rotate(x, x->pre->ch[0] == x);

/\*if(x->pre->ch[0] == x){

rotate(x, 1); //右旋

}else {

rotate(x, 0); //左旋

} \*/// <==> rotate(x, x->pre->ch[0] == x);

}else {

Node \*y = x->pre;

Node \*z = y->pre;

if(z->ch[0] == y){

if(y->ch[0] == x){

rotate(y, 1), rotate(x, 1);//zig-zig

}else{

rotate(x, 0), rotate(x, 1);//zig-zag

}//

}else {

if(y->ch[1] == x){

rotate(y, 0), rotate(x, 0);//zig-zig

}else {

rotate(x, 1), rotate(x, 0); //zig-zag

}

}

}

}

updata(x);

}

void select(int kth, Node \*f){

//找到第 k 个元素，并把该元素旋转使之成为 f 的儿子

int tmp;

Node \*t = root;

while(true){

tmp = t->ch[0]->sz;

if(tmp + 1 == kth){//find the right one

break;

}

if(kth <= tmp){

t = t->ch[0];

}else {

kth -= tmp + 1;

t = t->ch[1];

}

}

splay(t, f);

}

void insert(int pos, int val){

//在 pos 之前插入一个节点

Node \*tmproot = getNewNode(val);

select(pos - 1, null);

select(pos, root);

root->ch[1]->ch[0] = tmproot;

tmproot->pre = root->ch[1];

splay(root->ch[1], null);

}

void del(int k){

//删除第 k 个元素, k 不能为 1;

select(k, null);

Node \*old\_root = root;

root = root->ch[1];

root->pre = null;

select(1, null);

root->ch[0] = old\_root->ch[0];

root->ch[0]->pre = root;

updata(root);

S[++top] = old\_root;

}

void replace(int x, int y){

//把第 x 位置的元素值替换为 y

select(x, null);

root->val = y;

updata(root);

}

Node \*build(int l, int r){

// 这不就是线段树的建树过程吗？

if(l > r){

return null;

}

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

Node \*p = getNewNode(ary[m]);

p->ch[0] = build(l, m - 1);

if(p->ch[0] != null){

p->ch[0]->pre = p;

}

p->ch[1] = build(m + 1, r);

if(p->ch[1] != null){

p->ch[1]->pre = p;

}

updata(p);

return p;

}

};

Splay sp;

int main()

{

int n, m, i, x, y;

char c;

sp.init();

scanf("%d", &n);

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

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

}

Node \*troot = sp.build(1, n);

sp.root->ch[1]->ch[0] = troot;

troot->pre = sp.root->ch[1];

sp.updata(sp.root->ch[1]);

sp.updata(sp.root);

sp.splay(sp.root->ch[1], sp.null);

scanf("%d", &m);

for(int i = 1; i <= m; ++i){

scanf(" %c", &c);

if(c == 'I'){

scanf("%d %d", &x, &y);

sp.insert(++x, y);

}else if(c == 'D'){

scanf("%d", &x);

sp.del(++x);

}else if(c == 'R'){

scanf("%d %d", &x, &y);

sp.replace(++x, y);

}else if(c == 'Q'){

scanf("%d %d", &x, &y);

sp.select(++x - 1, sp.null);

sp.select(++y + 1, sp.root);

printf("%d\n", sp.root->ch[1]->ch[0]->mc);

}

}

return 0;

}