题意是一条线上的点，D x是破坏这个点，Q x是表示查询以x所在的最长的连续的点的个数，R是恢复上一次破坏的点。

线段树结点 设置一个 ll 记录区间左端点开始的最大连续个数， rl 记录区间右端点开始的最大的连续个数,

ml表示该区间最大的连续点的个数。

主要是更新和查询两个操作。

#include<iostream>

#include<cstdio>

#include<stack>

using namespace std;

//#define ll long long

struct node

{

int l,r;

int ll,rl,ml;

//左边开始连续的最大长度和右边开始最大的连续长度

//以及这个区间最大连续长度

}a[50010\*4];

inline void build(int root,int l,int r)

{

a[root].l=l;

a[root].r=r;

a[root].ll=a[root].rl=a[root].ml=r-l+1;

if(l==r)

return;

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

build(root<<1,l,mid);

build(root<<1|1,mid+1,r);

}

inline void update(int root,int x,int val)

{

if(a[root].l==a[root].r)//说明是叶子结点

{

if(val==1)

a[root].ll=a[root].rl=a[root].ml=1;

else

a[root].ll=a[root].rl=a[root].ml=0;

return;

}

int mid=(a[root].l+a[root].r)>>1;

if(x<=mid)

update(root<<1,x,val);

else

update(root<<1|1,x,val);

a[root].ll=a[root<<1].ll;

a[root].rl=a[root<<1|1].rl;

a[root].ml=max(max(a[root<<1].ml,a[root<<1|1].ml),a[root<<1].rl+a[root<<1|1].ll);

if(a[root<<1].ll==a[root<<1].r-a[root<<1].l+1)

a[root].ll+=a[root<<1|1].ll;

if(a[root<<1|1].rl==a[root<<1|1].r-a[root<<1|1].l+1)

a[root].rl+=a[root<<1].rl;

}

int query(int root,int x)

{

if(a[root].l==a[root].r || a[root].ml==0 || a[root].ml==a[root].r-a[root].l+1)

return a[root].ml;

int mid=(a[root].l+a[root].r)>>1;

if(x<=mid)

{

if(x>=a[root<<1].r-a[root<<1].rl+1)

return query(root<<1,x)+query(root<<1|1,mid+1);

else return query(root<<1,x);

}

else

{

if(x<=a[root<<1|1].l+a[root<<1|1].ll-1)

return query(root<<1|1,x)+query(root<<1,mid);

else return query(root<<1|1,x);

}

}

int main()

{

//freopen("input.txt","r",stdin);

int n,m;

while(~scanf("%d%d",&n,&m))

{

build(1,1,n);

char c[5];

stack<int>\_stack;

int x;

for(int i=0; i<m; ++i)

{

scanf("%s",c);

if(c[0]=='D')

{

scanf("%d",&x);

\_stack.push(x);

update(1,x,0);//0表示破坏

}

else if(c[0]=='Q')

{

scanf("%d",&x);

printf("%d\n",query(1,x));

}

else

{

x=\_stack.top();

\_stack.pop();

update(1,x,1);//1代表修复

}

}

}

return 0;

}