Hello World ! ---树状数组

/\*

树状数组 离线查询 好题

题意：给一序列{A}, 1 <= a[i] <= C. m个询问(l,r)，回答这个区间[l,r]出现了至少两次的数字。

思路：“HH的项链”加强版。

预处理:

离线存询问，按右端点排序。

记pre[i]为第i朵花前面，与i同色的最靠近i的花的位置。若不存在则pre[i]=0.

从左往右扫序列{A}，扫到i时，把树状数组中pre[pre[i]]+1 ~ pre[i]这一段上的元素+1:Update(pre[pre[i]]+1, 1), Update(pre[i]+1, -1);

然后询问以i为右端点的询问中，左端点在树状数组中的值即可:Query(que[j].l);

注意，回答的答案是区间[1, l]的和，注意这一点会比较好地理解为什么那样Update()

套路：存询问，按右端点排序，从左往右扫序列，遇到i == query[i].right就回答。

看题解好久才懂...=\_=

\*/

//bzoj 2743 heoi 2012

int main()

{

scanf("%d%d%d", &n, &C, &m);

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

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

pre[i] = head[a[i]];

head[a[i]] = i;

}

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

scanf("%d%d", &que[i].l, &que[i].r);

que[i].id = i;

}

sort(que, que+m, cmp);

int j = 0;

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

Update(pre[pre[i]]+1, 1), Update(pre[i]+1, -1);

while(j < m && que[j].r == i) {

ans[que[j].id] = Query(que[j].l); //

j++;

}

}

for(int i = 0; i < m; i++) printf("%d\n", ans[i]);

return 0;

}

/\*

二维树状数组

区间修改，询问单点

TODO : 到底“区间修改，询问单点” 和 “修改单点，询问区间” 有什么本质上的区别？ poj\_2155 与 poj\_1195

心得：先记录“区间修改，询问单点”的理解。

Update()和一般的树状数组的顺序倒了过来，因为Updtae(x,y,add)是指在[0,0,x,y]这个区间都加上add，所以我们就相当于标记区间一样，在[0,0,x,y]的“关键点”上加上add。为什么标记的时候是向下的顺序？理解了二维树状的构造就能大概理解了。

二维树状数组的构造:http://fqq11679.blog.hexun.com/21722866\_d.html

Query ()和一般的树状数组的顺序倒了过来，Query(x, y)就是查包含(x,y)这个点的所有区间的add之和。区间修改的时候是在区间上标记，则询问单点的时候就是向上询问包含(x,y)的区间标记之和。

那一般的“单点修改，区间求和”呢？这种情况是标记点，询问的时候向下询问，即向下把该区间的点的值都加起来，又称前缀和。

细节上面还没考虑，现在的水平还没办法思考细节，只能感性上理解。

\*/

void Update(int x, int y, int val)

{

for(int i = x; i > 0; i -= low(i))

for(int j = y; j > 0; j -= low(j))

c[i][j] += val;

}

int Query(int x, int y)

{

int result = 0;

for(int i = x; i <= n; i += low(i))

for(int j = y; j <= n; j += low(j))

result += c[i][j];

return result & 1;

}

int main()

{

int cases, x1,x2,y1,y2,query;

char op[5];

scanf("%d", &cases);

while(cases--) {

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

memset(c, false, sizeof(c));

while(query--) {

scanf("%s", op);

if(op[0] == 'Q') {

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

printf("%d\n", Query(x1, y1));

} else {

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

Update(x1-1, y1-1, 1); //TODO

Update(x2, y2, 1);

Update(x1-1, y2, -1);

Update(x2, y1-1, -1);

}

}

if(cases) printf("\n");

}

return 0;

}

/\*

二维树状数组 区间求和

修改单点，询问区间

\*/

void Update(int x, int y, int add)

{

for(int i = x; i <= n; i += low(i))

for(int j = y; j <= n; j += low(j)) c[i][j] += add;

}

int Query(int x, int y)

{

int sum = 0;

for(int i = x; i > 0; i -= low(i))

for(int j = y; j > 0; j -= low(j))

sum += c[i][j];

return sum;

}

int main()

{

int op,x,y,x1,x2,y2,y1,add;

scanf("%\*d%d", &n);

while(scanf("%d", &op), op != 3) {

if(op == 1) {

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

x++, y++;

Update(x, y, add);

} else {

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

x1++, y1++, x2++, y2++;

printf("%d\n", Query(x2,y2) + Query(x1-1,y1-1) - Query(x1-1, y2) - Query(x2, y1-1));

}

}

return 0;

}

/\*

理解好这道题的树状数组所代表的含义是解题的关键，可是网上的题解都没点出其树状数组的含义（难道是大家都觉得太简单了吗=\_=），然后我看题解都看不会，这两天切了几道简单的同类型题，有所收获。 反正我第一次想通这类树状数组的含义时是挺orz的...下面记录是建立在对树状数组的Update/Query过程已经有一个感性的理解的条件之上的。

Update(p, value) 的含义不是更新[1,p]这段区间的最大gcd值

相应地，c[p]的含义不是[1,p]这段区间的最大gcd值，Query(p)不是表示查询[1,p]这段区间的最大gcd值.

这几乎是“树状数组+离线询问”这类题目的共性了，这道题中的c[p]表示区间[1..p, i..∞]的最大gcd，即左端点为1到p，右端点为i到无穷大（其中i表示扫描到的当前位置，具体的还得看实现）。相应地Update(pos, value)在含义上应理解为更新[pos, i..∞]的最大gcd值（想一想，为什么）；要理解Update()为什么要赋予如此含义，还得理解好Query()的含义，这里Query(p)的返回值应为“[p, i]的最大gcd”。

基本上所有解法都是上述定义，不过可以有不同种实现方法。我最近就发现我做过的几道题把它们的Update/Query的方向倒过来，再改改插入/删除的操作，同样可以AC~这个看个人喜好。

树状数组+离线的大致套路：按右端点离线排序询问，从左往右扫序列，根据Update/Query的方向进行相应更新，遇到i == query[i].r 则算出 答案。c[p]表示区间[1..p, i..∞]或者[p, i..∞]的和/最值，即左端点为1到p，右端点为i到无穷大。pre[0]是堆拉圾的地方.

本题的解法网上已有很多，不赘述了，其实只要理解好树状数组的含义，再看题解，是很简单的。

\*/

void Update(int pos, int value)

{

while(pos) { //TODO 注意方向！！！

c[pos] = max(c[pos], value);

pos -= low(pos);

}

}

int Query(int pos)

{

int value = 0;

while(pos <= n) { //TODO 注意方向！！！

value = max(value, c[pos]);

pos += low(pos);

}

return value;

}

int main()

{

int cases;

get\_factor();

scanf("%d", &cases);

while(cases--) {

scanf("%d", &n);

init();

for(int i = 1; i <= n; i++) scanf("%d", &a[i]);

scanf("%d", &query);

for(int i = 0; i < query; i++) {

scanf("%d%d", &que[i].l, &que[i].r);

que[i].id = i;

}

sort(que, que+query, cmp);

int j = 0;

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

int & x = a[i];

for(int Size = factor[x].size(), k = 0; k < Size; k++) {

int fac = factor[x][k];

if(pre[fac] != 0) {

Update(pre[fac], fac);

}

pre[fac] = i;

}

while(j < query && que[j].r == i) {

ans[que[j].id] = Query(que[j].l);

j++;

}

}

for(int i = 0; i < query; i++) printf("%d\n", ans[i]);

}

return 0;

}

/\*

树状数组 + 离线查询

题意：给一个1-n的排列，问区间[l, r]里有多少段连续的数字。

思路：自己想了很久都没想出怎么维护...=\_=

看了大牛的一句话“我们可以维护一个数组数组或者线段树，对于每加进去的一个数num[i]，对 i 点加1，如果 i 点前存在num[i]-1和num[i]+1”恍然大悟...遂写，AC。

Update/Query的方向依然是跟一般线段树的方向反着的，我觉得这个方向比较好理解，所以就比较喜欢，虽说运行时间上比顺着慢...

\*/

void Update(int pos, int add)

{

while(pos) {

c[pos] += add;

pos -= low(pos);

}

}

int Query(int pos)

{

int sum = 0;

while(pos <= n) {

sum += c[pos];

pos += low(pos);

}

return sum;

}

int main()

{

int cases;

scanf("%d", &cases);

while(cases--) {

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

init();

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

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

pos[a[i]] = i;

}

for(int i = 0; i < query; i++) {

scanf("%d%d", &que[i].l, &que[i].r);

que[i].id = i;

}

sort(que, que+query, cmp);

int j = 0;

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

int x = a[i];

Update(i, 1);

if(pos[x-1] < i) Update(pos[x-1], -1);

if(pos[x+1] < i) Update(pos[x+1], -1);

while(j < query && que[j].r == i) {

ans[que[j].id] = Query(que[j].l);

j++;

}

}

for(int i = 0; i < query; i++) printf("%d\n", ans[i]);

}

return 0;

}

/\*

树状数组 离线查询 两个树状数组

题意：给出n个点的座标。不断定座标系，询问一三象限点数与二四象限点数的差。

思路：再写一遍套路：按右端点离线排序询问，从左往右扫序列，遇到i == query[i].r 则算出 答案。

总数不边，只需要算出一三象限的点数即可。

贮存询问，按x座标升序排好。为了方便，我把一般点和询问点都放一块儿，用一个label区分。所以这里升序排好的数组包括一般点和询问点。

建立两个树状数组c\_L[]表y轴右边的点，c\_R[]表y轴左边的点。初始时把所有点都加进c\_R[]里面。

开始扫，这里是直接根据点从左往右扫。遇到一般点，直接扔到c\_L[]里，并从c\_R[]里删除；遇到询问点，根据c\_L[]算出第三座标的点数，根据c\_R[]算出第一座标的点数。

因为题目说一般点不会有在轴上的情况，所以还是比较好处理的。

我做的时候还给y座标离散化了。

\*/

int main()

{

int cases;

scanf("%d", &cases);

while(cases--) {

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

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

scanf("%d%d", &a[i].x, &a[i].y);

a[i].label = NORMAL;

}

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

scanf("%d%d", &a[i].x, &a[i].y);

a[i].label = QUERY;

a[i].id = i - n;

}

memset(y\_map, 0, sizeof(y\_map));

sort(a, a+n+m, cmp\_y);

Index = 2;

y\_map[a[0].y] = 1;

for(int i = 1; i < n+m; i++) if(a[i].y != a[i-1].y) y\_map[a[i].y] = Index++; //离散化

sort(a, a+n+m, cmp\_x);

memset(c\_l, 0, sizeof(c\_l));

memset(c\_r, 0, sizeof(c\_r));

int tot = 0;

for(int i = 0; i < n+m; i++) if(a[i].label == NORMAL) Update(c\_r, y\_map[a[i].y], 1); //初始把所有点加进c\_L[]里

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

if(a[i].label == NORMAL) {

tot++;

Update(c\_l, y\_map[a[i].y], 1), Update(c\_r, y\_map[a[i].y], -1); //因为点不在轴上，所以还是比较好处理的。

} else {

int tmp = Query(c\_l, y\_map[a[i].y]) + (n - tot) - Query(c\_r, y\_map[a[i].y]); //一三象限的点数

ans[a[i].id] = tmp - (n - tmp);

}

}

for(int i = 0; i < m; i++) printf("%d\n", ABS(ans[i]));

if(cases) printf("\n");

}

return 0;

}