思路题

主讲人：邓哲也

HDU 5649 DZY Loves Sorting

DZY 有一个数组 a[1..n]，它是一个长度为 n 的全排列。

现在他想执行多次下列两种操作：

0 l r：表示对 a[l..r] 进行升序排序

1 l r：表示对 a[l..r] 进行降序排序

问经过 m 次操作后，a[k] 为多少？

Sample Input

```
1
6 3
1 6 2 5 3 4
0 1 4
1 3 6
0 2 4
3
```

Sample Output

```
5
```

HDU 5649 DZY Loves Sorting

如果每次对 [l, r] 真的执行一次排序，显然是不可能的。

用线段树的标记来完成对区间的排序，也是不现实的。

题目的突破口在于，询问在最后，且只询问一个位置上的数。

需要注意的是，m 次排序操作顺序不能乱，因为后面的排序是依赖于前面排好序的结果的。

HDU 5649 DZY Loves Sorting

考虑二分答案 x

那么排列中，大于等于 x 的数全都设为 1，小于 x 的数都设为 0.

现在区间排序，只要把区间里的 0 都移到最前面， 1 都移到最后面。

对于区间[l, r]，查询[l, r]的子段和 sum。

然后将[l, r - sum]修改为 0， [r - sum + 1, r]修改为 1.

HDU 5649 DZY Loves Sorting

如何判断 x 比答案大还是小？

如果最后 a[k] = 0，说明真实的答案小于 x。

如果最后 a[k] = 1

如果a[k - 1] = 0，说明答案就是 x

如果a[k - 1] = 1，说明真实的答案大于 x。

HDU 5649 DZY Loves Sorting

我们只要用线段树来实现：

区间求和

区间赋值

加上最外层的二分

总的时间复杂度为 $O(m \log^2 n)$

代码实现

```
#define N 200010
#define ls (x << 1)
#define rs (x << 1 | 1)
int tag[N << 2], sum[N << 2], n, m, a[N], op[N], l[N], r[N];

void down(int l, int r, int x){
    if (tag[x] != -1){
        int mid = (l + r) >> 1;
        sum[ls] = (mid - l + 1) * tag[x];
        sum[rs] = (r - mid) * tag[x];
        tag[ls] = tag[rs] = tag[x];
        tag[x] = -1;
    }
}
```

代码实现

```
void upd(int x){
    sum[x] = sum[ls] + sum[rs];
}

void build(int l, int r, int x, int v){
    tag[x] = -1;
    if (l == r){
        sum[x] = (a[l] > v);
        return;
    }
    int mid = (l + r) >> 1;
    build(l, mid, ls, v);
    build(mid + 1, r, rs, v);
    upd(x);
}
```

代码实现

```
void modify(int A, int B, int v, int l, int r, int x){
    if (A <= l && r <= B){
        sum[x] = v * (r - l + 1);
        tag[x] = v;
        return;
    }
    down(l, r, x);
    int mid = (l + r) >> 1;
    if (A <= mid) modify(A, B, v, l, mid, ls);
    if (mid < B)  modify(A, B, v, mid + 1, r, rs);
    upd(x);
}
```

代码实现

```
int query(int A, int B, int l, int r, int x){
    if (A <= l && r <= B) return sum[x];
    down(l, r, x);
    int mid = (l + r) >> 1, ret = 0;
    if (A <= mid) ret += query(A, B, l, mid, ls);
    if (mid < B)  ret += query(A, B, mid + 1, r, rs);
    return ret;
}
```

代码实现

```
int main(){
    int tc;
    scanf("%d", &tc);
    while(tc --){
        scanf("%d%d", &n, &m);
        for(int i = 1;i <= n;i ++) scanf("%d", &a[i]);
        for(int i = 1;i <= m;i ++) scanf("%d%d%d", &op[i], &l[i],
&r[i]);
        int K;
        scanf("%d", &K);
```

代码实现

```
        int L = 1, R = n;
        while (L <= R){
            int mid = (L + R) >> 1;
            build(1, n, 1, mid);
            for(int i = 1;i <= m;i ++){
                int c = query(l[i], r[i], 1, n, 1);
                modify(l[i], r[i], 0, 1, n, 1);
                if (op[i]){
                    if (c >= 1) modify(l[i], l[i] + c - 1, 1, 1, n, 1);
                } else {
                    if (c >= 1) modify(r[i] - c + 1, r[i], 1, 1, n, 1);
                }
            }
            if (query(K, K, 1, n, 1)) L = mid + 1; else R = mid - 1;
        }
        printf("%d\n", L);
    }
    return 0;
}
```

下节课再见