普通莫队

莫队，是莫涛发明的一种解决区间查询等问题的**离线**算法，基于分块思想，复杂度为 $O(n\sqrt{n})$ 。本文只涉及普通莫队。

一般来说，如果可以在 $O(1)$ 内从 $[l,r]$ 的答案转移到 $[l-1,r]$、$[l+1,r]$、$[l,r-1]$、$[l,r+1]$ 这四个与之紧邻的区间的答案，则可以考虑使用莫队。

转移基本上可以分为两类，一类是增加一个数（或者类似操作），另一类是减少一个数，我们一般用 `add` `del` 描述这两类操作：

```
void add(int p) {
    // do something
};

void del(int p) {
    // do something
};
```

那么，我们如果从区间 $[l_1,r_1]$ 移动到区间 $[l_2,r_2]$，则需要

```
while(L < Q[i].l) del(L++);
while(L > Q[i].l) add(--L);
while(R < Q[i].r) add(++R);
while(R > Q[i].r) del(R--);
```

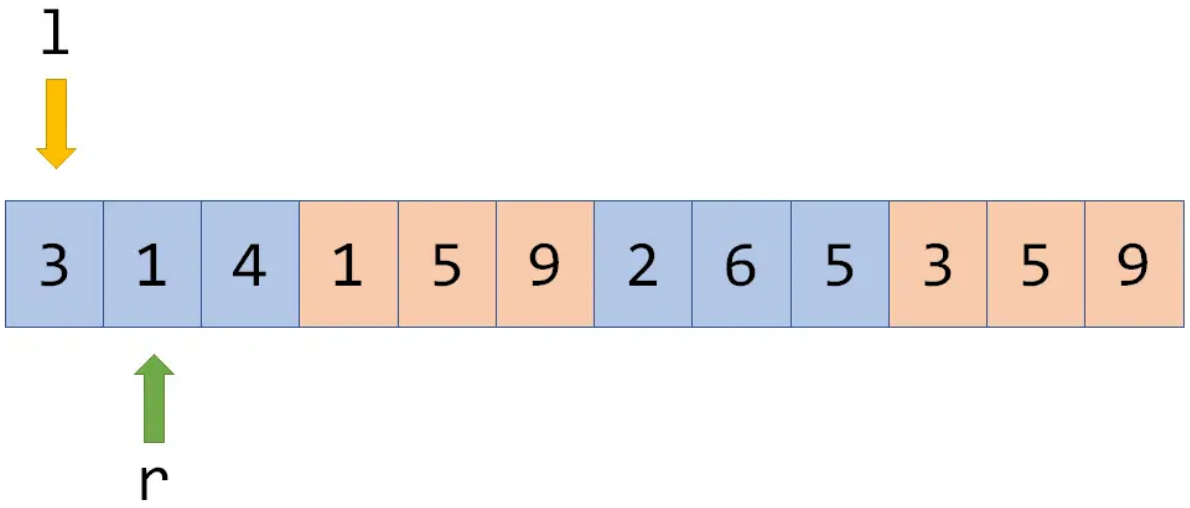
注意++和--的位置。**删数是先删后移，添数是先移后添。**初始化时，要先令$L=1,R=0$。（此处可以手动模拟一下）

现在我们可以从一个区间的答案转移到另一个区间了，但是，如果直接在线查询，很有可能在序列两头“左右横跳”，到头来还不如朴素的 $O(n^2)$ 算法。但是，我们可以把查询离线下来，然后进行排序

问题来了，怎么排序？我们很容易想到以 l 为第一关键词，r 为第二关键词排下序，但这样做效果并不是很好。莫涛大神给出的方法是，**分块**，然后按照 `l / sq` 为第一关键词，`r / sq` 为第二关键词排序。这样，每两次询问间 l 和 r 指针移动的距离可以被有效地降低，整个算法的时间复杂度可以降到 $O(n\sqrt{n}\times 1)$，其中，1 是`add` `del` 操作的事件复杂度，即如果你的操作不是 $O(1)$ 的，则要进行对应的计算

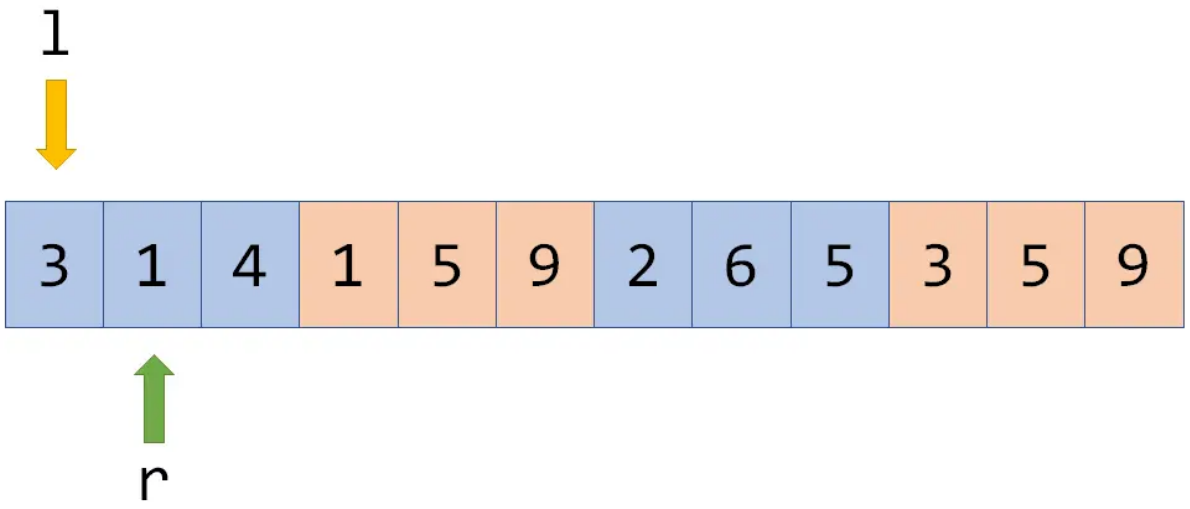
```
// struct Node { int l, r, id; };
// vector<Node> Q;
sort(Q.begin(), Q.end(), [&](const Node& a, const Node& b){
    if(a.l / sq != b.l / sq) return a.l < b.l;
    if((a.l / sq) & 1)return a.r < b.r;
    else return b.r < a.r;
});
```

但在此之上，我们还可以进行常数优化：**奇偶化排序。**意为：如果`l / sq`是奇数，则将`r`顺序排序，否则将`r`逆序排序。 这为什么有效？如果按照一般的排序方法，指针的动向可能是这样的：



我们看到，每次 l 跨过一个块时，r 都必须往左移很长一截。

而奇偶化排序后，指针的动向会变为这样：



可以发现，如果 l 在偶数块，r 指针会在返回的“途中”就解决问题。

拓展阅读

莫队算法——从入门到黑题