移动盒子

(move.cpp/.c)

限制：1S　256MB

题目描述：

一行有 n 个盒子，从左到右编号为 1～n。模拟以下 4 种命令。

- 1 X Y：将盒子 X 移动到 Y 的左侧（如果 X 已经在 Y 的左侧，则忽略此项）。
- 2 X Y：将盒子 X 移动到 Y 的右侧（如果 X 已经在 Y 的右侧，则忽略此项）。
- 3 X Y：交换盒子 X 和 Y 的位置。
- 4：翻转整行盒子序列。

以上命令保证有效，即 X 不等于 Y。

举例说明：有 6 个盒子，执行 1 1 4，即 1 移动到 4 的左侧，变成 2 3 1 4 5 6。然后执行 2 3 5，即 3 移动到 5 的右侧，变成 2 1 4 5 3 6。接着执行 3 1 6，即交换 1 和 6 的位置，变成 2 6 4 5 3 1。最后执行 4，即翻转整行序列，变成 1 3 5 4 6 2。

输入：(move.in)

最多有 10 个测试用例。每个测试用例的第 1 行都包含两个整数 n 和 m（1≤n, m≤100 000），下面的 m 行，每行都包含一个命令。

输出：(move.out)

对于每个测试用例，都单行输出奇数索引位置的数字总和。

输入样例	输出样例
6 4 1 1 4 2 3 5 3 1 6 4 6 3 1 1 4 2 3 5 3 1 6 100000 1 4	Case 1: 12 Case 2: 9 Case 3: 2500050000

题解

本题涉及大量移动元素，因此使用链表比较合适。但是将盒子 X 移动到盒子 Y 的左侧，还需要查找盒子 X 和盒子 Y 在链表中的位置，查找是链表不擅长的，每次查找的时间复杂度都为 O(n)，而链表的长度最多为 100 000，多次查找会超时，所以不能使用 list 链表实现。这里可以使用既具有链表特性又具有快速查找能力的静态链表实现，因为在题目中既有向前操作，也有向后操作，因此选择静态双向链表。另外，有大量元素的链表，其翻转操作的时间复杂度很高，会超时，此时只需做标记即可，不需要真的翻转。

1. 算法设计

（1）初始化双向静态链表（前驱数组为 l[]，后继数组为 r[]），翻转标记 flag=false。

（2）读入操作指令 a。

（3）如果 a=4，则标记翻转，flag=!flag，否则读入 x、y。

（4）如果 a!=3&&flag，则 a=3-a。因为如果翻转标记为真，则左右是倒置的，1、2 指令正好相反，即 1 号指令（将 x 移到 y 左侧）相当于 2 号指令（将 x 移到 y 右侧）。因此如果 a=1，则转换为 2；如果 a=2，则转换为 1。

（5）对于 1、2 指令，如果本来位置就是对的，则什么都不做。

（6）如果 a=1，则删除 x，将 x 插入 y 左侧。

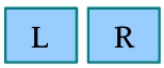
（7）如果 a=2，则删除 x，将 x 插入 y 右侧。

（8）如果 a=3，则考虑相邻和不相邻两种情况进行处理。

算法中的基本操作如下。

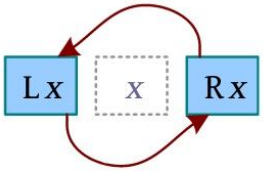
（1）链接。例如，将 L 和 R 链接起来，则 L 的后继为 R，R 的前驱为 L，如下图所示。

```
void link(int L,int R){//将 L 和 R 链接起来
     r[L]=R;
     l[R]=L;
}
```



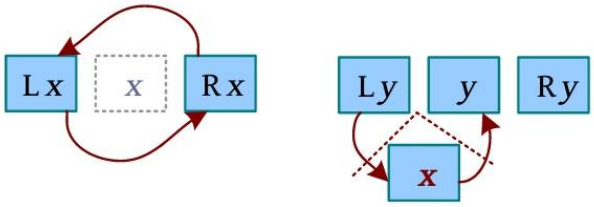
（2）删除。删除 x 时，只需将 x 跳过去，即将 x 的前驱和后继链接起来即可。

```
link(Lx,Rx);//删除 x
```

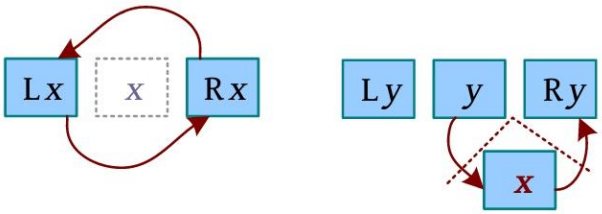


（3）插入（将 x 插入 y 左侧）。将 x 插入 y 左侧时，先删除 x，然后将 x 插入 y 左侧，删除操作需要 1 次链接，插入左侧操作需要两次链接，如下图所示。

```
link(Lx,Rx);//删除 x
link(Ly,x);//Ly 和 x 链接
link(x,y);//x 和 y 链接
```



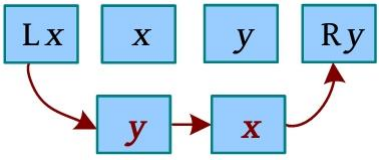
（4）插入（将 x 插入 y 右侧）。将 x 插入 y 右侧时，先删除 x，然后将 x 插入 y 右侧，删除操作需要 1 次链接，插入右侧操作需要两次链接，如下图所示。




```
link(Lx,Rx);//删除 x
link(y,x); //将 y 和 x 链接
link(x,Ry); //将 x 和 Ry 链接
```

（5）交换（相邻）。将 x 与 y 交换位置，如果 x 和 y 相邻且 x 在 y 右侧，则先交换、y，统一为 x 在 y 左侧处理。相邻情况的交换操作需要 3 次链接，如下图所示。

```
link(Lx,y); //Lx 和 y 链接
link(y,x); //y 和 x 链接
link(x,Ry); //x 和 Ry 链接
```

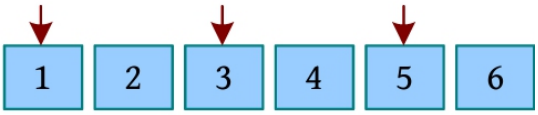


（6）交换（不相邻）。将 x 与 y 交换位置，如果 x 和 y 不相邻，则交换操作需要 4 次链接，如下图所示。

```
link(Lx,y); //Lx 和 y 链接
link(y,Rx); //y 和 Rx 链接
link(Ly,x); //Ly 和 x 链接
link(x,Ry); //x 和 Ry 链接
```

（7）翻转。如果标记了翻转，且长度 n 为奇数，则正向奇数位之和与反向奇数位之和是一样的。

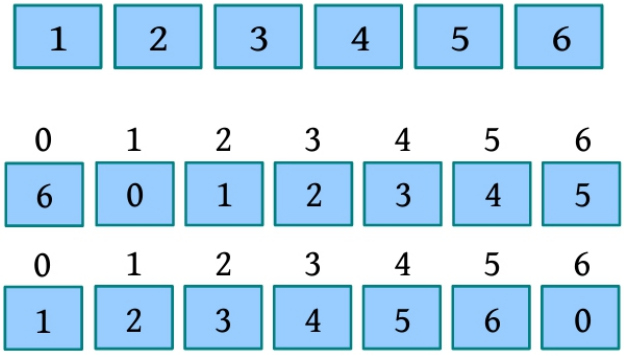
如果标记了翻转，且长度 n 为偶数，则反向奇数位之和等于所有元素之和减去正向奇数位之和。



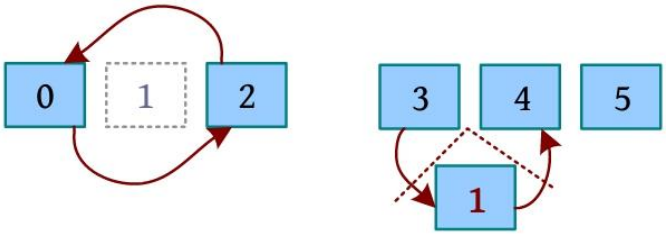
因此只需统计正向奇数位之和，再判断翻转标记和长度是否为偶数即可。

2. 完美图解

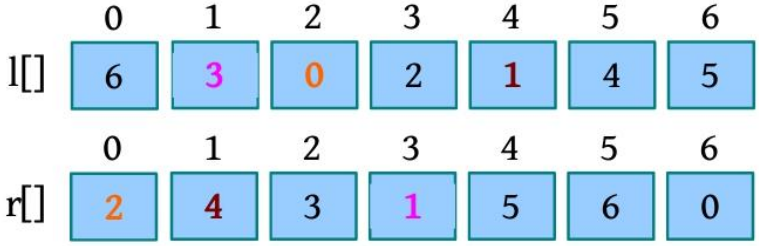
（1）以输入样例为例，n=6，初始化前驱数组和后继数组，如下图所示。



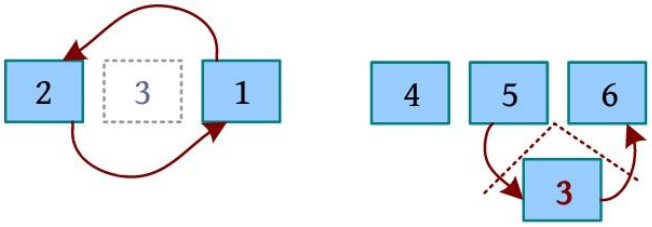
（2）1 1 4：执行 1 号指令（将 1 移到 4 左侧），先删除 1，然后将 1 插入 4 左侧。删除操作需要 1 次链接，插入需要两次链接，如下图所示。



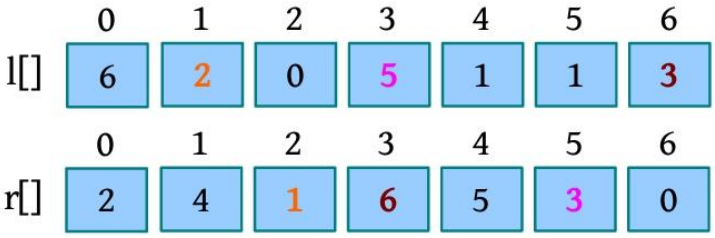
即修改 2 的前驱为 0，0 的后继为 2；1 的前驱为 1，3 的后继为 1；4 的前驱为 1，1 的后继为 4，如下图所示。



（3）2 3 5：执行 2 号指令（将 3 移到 5 右侧），先删除 3，然后将 3 插入 5 右侧。删除操作需要 1 次链接，插入需要两次链接，如下图所示。

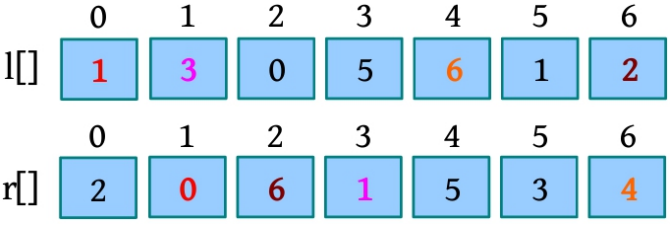


即修改 1 的前驱为 2，2 的后继为 1；3 的前驱为 5，5 的后继为 3；6 的前驱为 3，3 的后继为 6，如下图所示。



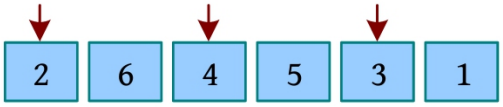
（4）3 1 6：执行交换（不相邻）指令，1 和 6 不相邻，交换操作需要 4 次链接。

即修改 4 个链接：6 的前驱为 2，2 的后继为 6；4 的前驱为 6，6 的后继为 4；1 的前驱为 3，3 的后继为 1；0 的前驱为 1，1 的后继为 0。



（5）4：执行翻转指令，标记翻转 flag=true。

（6）如果 n 为偶数且翻转为真，则反向奇数位之和等于所有数之和减去正向奇数位之和。



反向奇数位之和=所有数之和-正向奇数位之和=6×(6+1)/2-(2+4+3)=12。

3. 算法实现

[move.cpp]