**128MB,1S,.xxx**

异星工厂：信号网络

**问题描述**

你在异星工厂中设计了一个智能电力调度系统，当电池电量较低时，启用核电设备组，当更低时，启用煤电设备组。平时只用太阳能设备集群。又由于如果只设置单一标准线会导致一开启蓄电量就够了，一关闭，蓄电量就不够了，导致电力供应图一直疯狂振荡，很难看，需要更复杂的逻辑判断，开启和关闭设置不同的阈值，你在设计时，发现了位运算很有趣，于是你决定设计一个设备组，并用蓝图保存，来实现若干功能。

（ \*\* 提示中有大部份你需要掌握的知识，除非你很有基础，否则必看 \*\* ）

你需要维护一个128位无符号二进制数A，初始为0

我们将最低位（即$ 2^0$位）编号为0,最高位（即$ 2^127$位）编号为127,

支持

1 x 将A循环右移x位

2 查询A的后16位的十六进制表示,不需要输出0x前缀, 大写

3 l r 将A的第l位到第r位按位取反（或者说按位异或1）

**输入描述**

第一行一个整数N，表示操作次数

接下来N行每行1到3个整数，格式见问题描述

$ 0 <= N <= 2\times 10^6; |x| <= 10^9; 0 <= l <= r <= 127 $

x < 0 表示向左循环位移 $|x|$位

**输出描述**

对于每个2操作，输出一个16进制数，不需要输出0x前缀，大写

**样例输入**

**样例输出**

**数据范围及提示**

\_\_uint128\_t 表示128位无符号整数

位运算的知识：施工中

我本来想的是要求你用两个128位整数拼成一个256位整数，但考虑到难度，就放弃了

由题目中的要求，并不要求数有正负之分，所以可以用无符号整数表示，

而且有循环右移的要求，所以需要用无符号整数来表示（因为对于有符号整数，最高位为符号位，右移时，会补充符号位）

我们以8位二进制整数为例，讲解一些运算

最高位在左，最低位在右（由于存在大端机和小端机的区别，导致其物理存储顺序不同，但仅考虑位运算时，你并不需要区分）

比如 $ 5 == 2^2 + 2^0 $可以表示为

00000101

其右移1位即（移出的位会被省略）

00000010

其左移1位即

00001010

其左移6位即 （移出的位会被省略）

10000000

那么如何表示负数呢，在C以及绝大部分计算机应用中，使用补码方式，

简单理解就是：比如x是一个非负整数，那么-x就是将x的二进制取反，然后+1

比如1的二进制为00000001

按位取反（即每一位翻转）

11111110

+1

11111111

比如2

00000010

按位取反

11111101

+1

11111110

比如0

00000000

按位取反

11111111

+1

00000000 （进位溢出会被省略）

特别得如果这是一个有符号整数，且最高位（即符号位）为1

比如-2

11111110

其右移1位并不是

01111111

而是

11111111

因为其前方因为右移移出的空位（在左侧）会被替换为符号位（在这里为1）

比如-128其为

10000000

右移2位为

11100000

整数的右移，是指将所有位向右可以看作除以2下取整比如, 1 >> 1 == 0, 0 >> 1 == 0, 2 >> 1 == 1, -3 >> 1 == -2, -1 >> 1 == -1

循环右移指的是将向右溢出的字符补充到左侧空出来得位置，比如

00011101 循环右移 1位

结果为

10001110

00011101 循环右移 3位

10100011

我们发现当$ x\_{7~0}$循环右移y(1 <= y <= 7)位时，其变成了$x\_{y-1 ~ 0, 7 ~ y}$

若y >= 8 或y < 0 可以令y对8取模，（y = (y % 8 + 8) % 8; ）

如果y == 0相当于不变（当然，你不一定要特判y == 0，好的实现方法可以兼容这种情况，如果不够好，可以特判一下）

理论上你知道这个性质，再加上或运算就可以得到循环位移的写法了

如果想不出，可以百度“c++循环右移”

与，或，异或，按位取反

与：a & b

对于每一位若两者均为1,则为1,否则为0

比如

11001101 & 00110101

==

00000101

或：a | b

对于每一位若两者均为0,则为0,否则为1

11001101 | 00110101

==

11111101

异或：a ^ b

对于每一位若两者相同,则为0,否则为1

11001101 ^ 00110101

==

11111000

按位取反: ~a

~11001101

==

00110010

当(a & b) == 0时

(a ^ b) == (a | b)

这里加括号是为了运算符优先级

注意区分& 和 &&; | 和 ||

我们如何取出低i位呢

我们令mask(i) = 表示低i位均为1,其余均为0

比如（这里的=是数学上的等于）mask(5) = 00011111 = $2^5 - 1$ = (1 << 5) - 1

特别的，对于8位无符号整数mask(8) = 11111111 = ~0 = -1 $\not=$ (1 << 8) - 1

但是在实际运算时，不足32位的整数在运算时会自动转化为32位的，所以中在C可以用(1 << 8) - 1

但对于 >= 32位的整数，比如128位整数，不能用 ((\_\_uint128\_t)1 << 128) - 1来表示全1

不考虑自动类型转化的话，继续以8位无符号整数举例

1 << (8 + 1) 和 1 << 8 << 1 是不等价的

因为位移时，比如a << b; 只考虑b的低三位，即b mod 8的值

故1 << 8 == 1 << 0

在b为负数时同理，a >> -2 即为 a >> (8 - 2)

a << -2 即为 a << (8 - 2)

回到mask(i)

a mod $ 2^i $ = a & mask[i] ;//a为负数时也成立，这里的mod是数学意义上的mod，已经取正

mask(l) ^ mask(r + 1)即第l位到第r位均为1,其余为0

这里的思想称为前缀和（当然，不一定是前缀，可以是后缀，和也不一定是数的相加）