基础数据类型中二进制位操作算法

大家好，欢迎大家来到Coding迪斯尼。程序在运行过程中，会根据指令更新在内存中的变量数值。而程序的变量，可以根据它的类型来划分。类型决定了变量可取值的范围以及什么样的运算可以发生在该变量上。变量的类型，有一大类，我们称之为基础数据类型，在 C++中，基础类型有: bool, char, short, int, long, float, double. 在 java 中有 boolean, char, byte, short, int, long, float, double. 很多复杂的类型基本上都是各种基础类型的组合。

在面试算法题中，有很大一块涉及到对基础数据类型二进制层级的操作。一个此方面经典的面试题是这样的：给定两个整形变量a,b.交换他俩的值可以这么做, temp = a; a = b; b = temp. 问题是，你是否能直接交换a,b的值而不使用其他变量。事实上是可以的：a = a ^ b; b = a ^ b; a = a ^ b; 其中^表示亦或操作。更简单的写法是: a ^= b ^=a ^=b; 这个问题在我的面试经历中，被问过不下三次，愚蠢的我都没答对，大家在此留个心眼，一旦面试中遇到，可别答错。

二进制层级的算法内容，主要涉及到位的运算与操作。很多关于二进制层级的算法设计都要基于一些基础操作之上，下

面我们看看二进制位的基础操作都有哪些：

x & (x-1)，该操作是把 x 的二进制表示中的最低位的 1 给清零，例如 x= 0b1010110 执行上面的语句后 x 变成 0b1010100

x & !(x-1) 得到最低位的 1,例如 x = 0b1010110，x - 1 就等于 0b1010101，!(x-1) 就等于 0b0101010，x & !(x-1) 就等于 0b0000010.于是 x 二进制表示中，最低位的 1 我们就取到了。

二进制操作中，有一个很重要的操作是交换第 i 和 第 j 位的值(最低位的起始从 0 开始)，例如 x = 0b0101，i = 0，j = 1，交换 i 和 j 两处的位后 x 变为 0b0110.该操作的实现如下：

```
int swap_bit(int x, int i, int j) {
    if ( ( (x >> i) & 1) != ( (x>>j) & 1) ) {
        x ^= (1 <<i) | (1 <<j);
    }

    return x;
}
```

(x >>i) & 1 获取 x 二进制表示的第 i 位的值，(x>>j)&1 获取 x 二进制第 j 位的值，如果 i 和 j 两处的二进制数值相同，那么就没有交换的必要，如果不同，x ^= (1<<i)｜(1 << j) 就把 x 的 i 和 j 两处的二进制数值互换，举个例子, x = 0b101010 , i = 1， j = 2, 那么 (1 <<j) ｜ (1<<i)的值就是 0b000110, 两者做亦或操作:

```
0b 1 0 1 0 1 0
 ^
0b 0 0 0 1 1 0     =

0b 1 0 1 1 0 0
```

也就是第 i 位和第 j 位的值成功互换了。

二进制操作中，还有一种重要的基础操作就是将 64 位整形数的二进制的表示形式倒转，例如 x = 0b111000 倒转后变成 0b000111，基于上面讲过的位互换操作，我们很容易实现该功能:

```
int  reverse_bit(int x) {
    int  j = Integer.size - 1, i = 0;
    while (i < j) {
        x = swap_bit(x, i, j);
        j--;
        i++;
    }

    return x;
}
```

基于上面所说的二进制基础操作，我们接下来深入分析有关位操作的算法面试题。

对于 64 或 32 位的无符号整形数 x，我们把它的二进制表示中，1 的个数称为 x 的权重，例如 x = 7 ， 它的二进制表示为 111,那么 x 的权重就是 3. 用 S(k)表示 64 或 32 位无符号整形数中，权重是 k 的所有整数的集合，其中 k 不等于 0 和 64，给定一个整形数 x，x 属于 S(k)，要求你找到另一个整形 y 属于 S(k)，y 不等于 x，并且使得 | x - y| 的值最小。

大家此时可以把视频暂停来思考一下这个问题。

大家是否还记得前几节我们讲过的小数量数据分析模式，也就是拿一些简单的实际数据放入到题目中检测一下，看看有没有规律可寻。在此，我们也试试一些具体数据，例如我们假设 k = 3, 也就是整数的二进制形式含有 3 个 1，不妨设 x = 0b1011, 我们取一些 y 值计算一下，例如 y = 0b01110, 那么 |x - y| = 0b101, 如果 y = 0b1101, |x - y| = 0b10, 如果 y = 0b111, |x - y| = 0b100, 通过一系列测试，我们可发现当 x=0b1011，满足条件的 y=0b1101. 我们再测试一下其他情况，假设 k=4, x = 0b11011, 如果 y=0b11110, |x-y|=0b11, 如果 y = 0b11101, |x-y|=0b10, 如果 y=0b10111, |x-y|=0b100, 经过一系列测试我们发现，当 x=0b11011, y=0b11101 时，可满足条件。我们把两次测试的结果放在一起看看：

1.

x = 0b1011

y = 0b1101

2.

x = 0b11011

y = 0b11101

其中我们可以发现一个规律是，满足条件的 y,似乎是对 x 的二进制形式中，从最右边开始往左走，当第一次发现相邻两位的值不同时，交互这两位，就可以得到满足条件的 y，例如 x=0b1011，从右向左，第一次出现两位值不同的下标是 1 和 2，因此只要把第 1 和 2 两位互换，也就得到 0b1101。如果 x=0b11011,依照相同的做法，我们也可以得到上面所说的最优的 y 值。于是乎我们得到了一个可行的算法，也就是从最低位，也就是从最右边向左边遍历，当第一次发现相邻两位的数值不同时，交换不同的这两位，所得的结果就是满足题目要求的 y 值。

我们从理论上证明上面做法的正确性,假定 x，y 都属于 S(k)，如果 x 二进制的第 i 位上是 1，第 j 位上是 0，而 y 正好相反,第 j 位上是 1,第 i 位上是 0,那么 |x-y| = |2^i - 2^j|，要想使得|x-y|最小，必须满足两个条件：

1. x，y 的二进制表示只有在 i，j 两处不同，其他处都相同，要不然，假设 x 的第 k 位是 0，l 位是 1，而 y 正好相反，那么|x-y| = |2^i - 2^j + 2^k - 2^l|，这样，

两者的差值就变大了。

2. i 和 j 的距离要尽可能的接近，最接近的情况就是 i 和 j 相邻。

我们上面提出的算法满足这两条，所以算法的正确性可以保证，由于算法要遍历一次 x 的二进制表示，假设 x 的二进制由 n 位构成，那么算法的时间复杂度就是 O(n). 以下就是我们的代码实现:

```
int   find_min_abs(int x) {
    int i = 0;
    while ( i < Integer.size) {
        if (swap_bit(x, i, i + 1) != x) {
            return swap_bit(x, i, i + 1);
        }
        i++;
    }
}
```

代码实现中，我们使用到了前面提到的位交互基础操作。

总结：

本章我们进入了基础数据类型算法方面的研究，我们讨论了涉及基础数据类型二进制表现形式的若干基础操作，这些基础操作分别是把最低位的 1 清零，获取二进制表示中最低位的 1，交换指定位置的数据位，以及实现二进制表现形式的倒转。最后，我们分析了一道涉及二进制操作的面试算法题，并给出了算法设计和代码实现。

大家在理解了上面的内容后，一定要用纸和笔，在不看资料的情况下，将本章所描述的算法重新实现一边。因为大多数面试都是通过笔纸或白板的方式进行的，所以大家用笔和纸来写算法代码才能更好的模拟实际面试情况。