位运算

每一个整数都是以二进制的形式存储
how does computer store an integer

- Position : 31　　———>　　0 (high to low)
- Binary : (0 0 0 0 0... 1 1 1 1 0 1 0 1 0 1 1 0)
- decimal: 2^1 + 2^2 + 2^4 + 2^6 + 2^8 + 2^9 + 2^10 + 2^11
- (最高位是符号位) The highest bit is the sign bit
- 0 : positive number
- 1: negative number

(正数)Positive number

- 1 sign bit，31 value bit

- 1个符号位，31个数值位

- 有符号32位最大的正整数(biggest signed integer of 32 bits)
- (0 1 1 1 1 1 1 1 ... 1 1 1 1 1 1 1 1) = 2147483647

负数(negative number)

- -2147483648 = (1000 0000 0000 … 0000)
- -2147483647 = (1000 0000 0000 … 0001)
- -2 = (1 1 1 1 1… 1 1 1 1 1 1 1 1 1 1 1 0)
- -1 = (1 1 1 1 1… 1 1 1 1 1 1 1 1 1 1 1 1)
- -1 is the biggest negative number

magic

- 3 - 2 = 3 + (-2)
- 0 0 0 0 0... 0 0 0 0 0 0 0 0 0 0 1 1 (3)
- 1 1 1 1 1... 1 1 1 1 1 1 1 1 1 1 1 0 (-2)
- 0 0 0 0 0... 0 0 0 0 0 0 0 0 0 0 0 1 (1)
- 通过合理的设计负数的表示方法，我们只需要一个加法器就能让计算机同时实现正数与负数的运算
- 对于计算机本身而言，它并没有正负之分，它只是将一串串的01码进行了假发运算而已
- 人类通过合理的设计将运算对应到了我们能理解的方式

why

- 将八点钟调到五点钟可以逆时针调节三小时
- 也可以顺时针调节12 - 3 = 9小时
- 取模!
- -2的原码是(100…00…010)
- 原码取反再加1即为补码，补码就是顺时针拨时针的数量
- 所以负数以补码的形式保存就相当于调节时针的原理
- 最大的数再加1就成了最小的数
- 所有的数构成了一个环，循环往复

与& (and)

- 7 & 6 = 6
- 0111
- 0110
- 0110

或| (or)

- 3 | 6 = 7

- 011
- 110
- 111

异或^ (xor)

- 7 ^ 6 = 1
- 111
- 110

左移 << (left shift)

- 4 << 1 = 8
- 1 << 10 = 1024

右移>>(right shift)

- 4 >> 1 = 2
- 4 >> 2 = 1
- 6 >> 2 = 1

输出一个数的二进制表示
output a number in binary

```
int x = 123456789;
for (int i = 31; i >= 0; i--)   {
    if(x & (1LL << i)) {
        cout << 1;
    } else {
        cout << 0;
    }
}
```

枚举n个数的所有组合
enumerate all the combination of n numbers

```
int a[] = {1, 2, 3, 4, 5};
for (int i = 0; i < (1 << n); i++) {
    for (int j = 0; j < n; j++) if(i & (1 << j)) {
        cout << a[j] << " " ;
    }
    cout << endl;
}
```

快速冪(fast pow)

- 求 a ^ b % c
- 1 <= b <= 10^9

- a^13%c
- a*a%c=(a%c*a%c)%c
- 13 = (1101) = 2^3 + 2^2 + 2^0
- 1 1 0 1
- a^8 a^4 a^2 a

```
int pow_mod(int a, int b, int c) {
    int ret = 1;
    while (b > 0) {
        if (b % 2 == 1) {
            ret = 1LL * ret * a % c;
        }
        b = b / 2;
        a = 1LL * a * a % c;
    }
    return ret%c;
}
```