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码进行了加法运算而已
- 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;
}
```

- 一个整数的二进制可以表示一个状态
- 10111
- 表示选择了第一个，第二个，第三个，第五个
- 子结构就是那些1的位置是它的子集的数，比如10011，10001

- 去掉某一个位置可以表示为mask ^ (1 << j)
- mask | (1 << j) 合并某一个位置的信息 到达新状态
- 判断二进制的某一位是否是1， mask >> j & 1

汉密尔顿通路

- 给你一副图，求从某个点出发经过每个点恰好一次的最短路
- dp[1 << n][n]
- dp[mask][i]表示经过了mask状态的点，停留在i号点的最短路
- 转移的时候枚举i的邻接点即可
- 初始状态dp[1 << i] [i] = 0,表示枚举每个出发点
- https://vjudge.net/problem/LightOJ-1316

麦当娜的梦想poj 2411

- 用1*2的砖块去铺满n*m的网格，有几种拼法
- dp[i][mask]表示前i行已经铺满， 第i+1行的状态为mask
- 暴力dfs去将mask填满然后得到新的下一行的状态去转移
- 细节较多，有些繁琐
- http://poj.org/problem?id=2411

		0	0
0	1		

- 考虑逐格转移，状态是一条轮廓线，轮廓线上方的格子已经铺满了，轮廓线上的状态用滚动数组记录，转移的时候首先要将上方的那个红色格子填充掉，只需要考虑当前格子的左边与上方格子的情况就可以了
-

SRM 667 div2 T2

- 给你一个20*20的01矩阵，你可以任意的选择行的顺序，假如当前选择的行上有K个1位置所在的列都是新增的，这一次选择这一行的代价就是k*k,你需要合理的安排行被选择的顺序，是的总代价最小

- 111
- 101
- 010
- 答案是5

- dp[mask]表示选了mask的状态的行花费的最小代价，枚举一个没有选择过的行去转移到一个新状态，每一个状态都对应着一个列的覆盖情况，需要记录下来

- dp[(1<<n)-1]就是答案

lowbit

- 由二进制的基础知识可以得到
- x & -x 可以计算出x的最后一个1位代表的二的幂次
- 利用这个我们可以在转移的时候快速的遍历一个二进制所有的1位或者0位，优化不少常数

srm 449div1 T2

- 给你一个蜂窝形状的图形，有一些格子已经被占据，求将剩下的格子用1*2的砖块尽可能的铺满的总方案数

DNA序列

- https://vjudge.net/problem/LightOJ-1073