位运算简介及实用技巧（一）：基础篇

去年年底写的关于位运算的日志是这个 Blog 里少数大受欢迎的文章之一，很多人都希望我能不断完善那篇文章。后来我看到了不少其它的资料，学习到了更多关于位运算的知识，有了重新整理位运算技巧的想法。从今天起我就开始写这一系列位运算讲解文章，与其说是原来那篇文章的 follow-up，不如说是一个 remake。当然首先我还是从最基础的东西说起。

什么是位运算？

程序中的所有数在计算机内存中都是以二进制的形式储存的。位运算说穿了，就是直接对整数在内存中的二进制位进行操作。比如，and 运算本来是一个逻辑运算符，但整数与整数之间也可以进行 and 运算。举个例子，6 的二进制是 110，11 的二进制是 1011，那么 6 and 11 的结果就是 2，它是二进制对应位进行逻辑运算的结果（0 表示 False，1 表示 True，空位都当 0 处理）：

```
   110
AND 1011
  ----------
   0010  --> 2
```

由于位运算直接对内存数据进行操作，不需要转成十进制，因此处理速度非常快。当然有人会说，这个快了有什么用，计算 6 and 11 没有什么实际意义啊。这一系列的文章就将告诉你，位运算到底可以干什么，有些什么经典应用，以及如何用位运算优化你的程序。

Pascal 和 C 中的位运算符号

下面的 a 和 b 都是整数类型，则：

```
C 语言  |  Pascal 语言
-------+-------------
a & b  |  a and b
a | b  |  a or b
a ^ b  |  a xor b
 ~a    |    not a
a << b |  a shl b
a >> b |  a shr b
```

注意 C 中的逻辑运算和位运算符号是不同的。520|1314=1834，但 520||1314=1，因为逻辑运算时 520 和 1314 都相当于 True。同样的，!a 和~a 也是有区别的。

各种位运算的使用

=== 1. and 运算 ===

and 运算通常用于二进制取位操作，例如一个数 and 1 的结果就是取二进制的最末位。这可以用来判断一个整数的奇偶，二进制的最末位为 0 表示该数为偶数，最末位为 1 表示该数为奇数.

=== 2. or 运算 ===

or 运算通常用于二进制特定位上的无条件赋值，例如一个数 or 1 的结果就是把二进制最末位强行变成 1。如果需要把二进制最末位变成 0，对这个数 or 1 之后再减一就可以了，其实际意义就是把这个数强行变成最接近的偶数。

=== 3. xor 运算 ===

xor 运算通常用于对二进制的特定一位进行取反操作，因为异或可以这样定义：0 和 1 异或 0 都不变，异或 1 则取反。

xor 运算的逆运算是它本身，也就是说两次异或同一个数最后结果不变，即(a xor b) xor b = a。xor 运算可以用于简单的加密，比如我想对我 MM 说 1314520，但怕别人知道，于是双方约定拿我的生日 19880516 作为密钥。1314520 xor 19880516 = 20665500，我就把 20665500 告诉 MM。MM 再次计算 20665500 xor 19880516 的值，得到 1314520，于是她就明白了我的企图。

下面我们看另外一个东西。定义两个符号#和@（我怎么找不到那个圈里有个叉的字符），这两个符号互为逆运算，也就是说(x # y) @ y = x。现在依次执行下面三条命令，结果是什么？

```
x <- x # y
y <- x @ y
x <- x @ y
```

执行了第一句后 x 变成了 x # y。那么第二句实质就是 y <- x # y @ y，由于#和@互为逆运算，那么此时的 y 变成了原来的 x。第三句中 x 实际上被赋值为(x # y) @ x，如果#运算具有交换律，那么赋值后 x 就变成最初的 y 了。这三句话的结果是，x 和 y 的位置互换了。

加法和减法互为逆运算，并且加法满足交换律。把#换成+，把@换成-，我们可以写出一个不需要临时变量的 swap 过程(Pascal)。

```
procedure swap(var a,b:longint);
begin
     a:=a + b;
     b:=a - b;
     a:=a - b;
end;
```

好了，刚才不是说 xor 的逆运算是它本身吗？于是我们就有了一个看起来非常诡异的 swap 过程：

```
procedure swap(var a,b:longint);
begin
     a:=a xor b;
     b:=a xor b;
     a:=a xor b;
end;//a 与 b 不得为 0
```

=== 4. not 运算 ===

not 运算的定义是把内存中的 0 和 1 全部取反。使用 not 运算时要格外小心，你需要注意整数类型有没有符号。如果 not 的对象是无符号整数（不能表示负数），那么得到的值就是它与该类型上界的差，因为无符号类型的数是用$0000 到$FFFF 依次表示的。下面的两个程序（仅语言不同）均返回 65435。

```
var
     a:word;
```

```
begin
     a:=100;
     a:=not a;
     writeln(a);
end.
#include <stdio.h>
int main()
{
        unsigned short a=100;
        a = ~a;
        printf( "%d\n", a );
        return 0;
}
```

如果 not 的对象是有符号的整数，情况就不一样了，稍后我们会在“整数类型的储存”小节中提到。

=== 5. shl 运算 ===

a shl b 就表示把 a 转为二进制后左移 b 位(在后面添 b 个 0)。例如 100 的二进制为 1100100，而 110010000 转成十进制是 400，那么 100 shl 2 = 400。可以看出，a shl b 的值实际上就是 a 乘以 2 的 b 次方，因为在二进制数后添一个 0 就相当于该数乘以 2。

通常认为 a shl 1 比 a * 2 更快，因为前者是更底层一些的操作。因此程序中乘以 2 的操作请尽量用左移一位来代替。

定义一些常量可能会用到 shl 运算。你可以方便地用 1 shl 16 - 1 来表示 65535。很多算法和数据结构要求数据规模必须是 2 的幂，此时可以用 shl 来定义 Max_N 等常量。

=== 6. shr 运算 ===

和 shl 相似，a shr b 表示二进制右移 b 位（去掉末 b 位），相当于 a 除以 2 的 b 次方（取整）。我们也经常用 shr 1 来代替 div 2，比如二分查找、堆的插入操作等等。想办法用 shr 代替除法运算可以使程序效率大大提高。最大公约数的二进制算法用除以 2 操作来代替慢得出奇的 mod 运算，效率可以提高 60%。

位运算的简单应用

有时我们的程序需要一个规模不大的 Hash 表来记录状态。比如，做数独时我们需要 27 个 Hash 表来统计每一行、每一列和每一个小九宫格里已经有哪些数了。此时，我们可以用 27 个小于 2^9 的整数进行记录。例如，一个只填了 2 和 5 的小九宫格就用数字 18 表示（二进制为 000010010），而某一行的状态为 511 则表示这一行已经填满。需要改变状态时我们不需要把这个数转成二进制修改后再转回去，而是直接进行位操作。在搜索时，把状态表示成整数可以更好地进行判重等操作。这道题是在搜索中使用位运算加速的经典例子。以后我们会看到更多的例子。

下面列举了一些常见的二进制位的变换操作。

功能	示例	位运算

去掉最后一位 | (101101->10110) | x shr 1
在最后加一个 0 | (101101->1011010) | x shl 1
在最后加一个 1 | (101101->1011011) | x shl 1+1
把最后一位变成 1 | (101100->101101) | x or 1
把最后一位变成 0 | (101101->101100) | x or 1-1
最后一位取反 | (101101->101100) | x xor 1
把右数第 k 位变成 1 | (101001->101101,k=3) | x or (1 shl (k-1))
把右数第 k 位变成 0 | (101101->101001,k=3) | x and not (1 shl (k-1))
右数第 k 位取反 | (101001->101101,k=3) | x xor (1 shl (k-1))
取末三位 | (1101101->101) | x and 7
取末 k 位 | (1101101->1101,k=5) | x and (1 shl k-1)
取右数第 k 位 | (1101101->1,k=4) | x shr (k-1) and 1
把末 k 位变成 1 | (101001->101111,k=4) | x or (1 shl k-1)
末 k 位取反 | (101001->100110,k=4) | x xor (1 shl k-1)
把右边连续的 1 变成 0 | (100101111->100100000) | x and (x+1)
把右起第一个 0 变成 1 | (100101111->100111111) | x or (x+1)
把右边连续的 0 变成 1 | (11011000->11011111) | x or (x-1)
取右边连续的 1 | (100101111->1111) | (x xor (x+1)) shr 1
去掉右起第一个 1 的左边 | (100101000->1000) | x and (x xor (x-1))

最后这一个在树状数组中会用到。

Pascal 和 C 中的 16 进制表示

Pascal 中需要在 16 进制数前加$符号表示，C 中需要在前面加 0x 来表示。这个以后我们会经常用到。

整数类型的储存

我们前面所说的位运算都没有涉及负数，都假设这些运算是在 unsigned/word 类型（只能表示正数的整型）上进行操作。但计算机如何处理有正负符号的整数类型呢？下面两个程序都是考察 16 位整数的储存方式（只是语言不同）。

```
var
     a,b:integer;
begin
     a:=$0000;
     b:=$0001;
     write(a,' ',b,' ');
     a:=$FFFE;
     b:=$FFFF;
     write(a,' ',b,' ');
     a:=$7FFF;
     b:=$8000;
     writeln(a,' ',b);
end.
```

```
#include <stdio.h>
int main()
{
        short int a, b;
        a = 0x0000;
        b = 0x0001;
        printf( "%d %d ", a, b );
        a = 0xFFFE;
        b = 0xFFFF;
        printf( "%d %d ", a, b );
        a = 0x7FFF;
        b = 0x8000;
        printf( "%d %d\n", a, b );
        return 0;
}
```

两个程序的输出均为 0 1 -2 -1 32767 -32768。其中前两个数是内存值最小的时候，中间两个数则是内存值最大的时候，最后输出的两个数是正数与负数的分界处。由此你可以清楚地看到计算机是如何储存一个整数的：计算机用$0000 到$7FFF 依次表示 0 到 32767 的数，剩下的$8000 到$FFFF 依次表示-32768 到-1 的数。32 位有符号整数的储存方式也是类似的。稍加注意你会发现，二进制的第一位是用来表示正负号的，0 表示正，1 表示负。这里有一个问题：0 本来既不是正数，也不是负数，但它占用了$0000 的位置，因此有符号的整数类型范围中正数个数比负数少一个。对一个有符号的数进行 not 运算后，最高位的变化将导致正负颠倒，并且数的绝对值会差 1。也就是说，not a 实际上等于-a-1。这种整数储存方式叫做“补码”。

位运算简介及实用技巧（二）：进阶篇(1)

=====　　真正强的东西来了！　　=====

二进制中的 1 有奇数个还是偶数个

我们可以用下面的代码来计算一个 32 位整数的二进制中 1 的个数的奇偶性，当输入数据的二进制表示里有偶数个数字 1 时程序输出 0，有奇数个则输出 1。例如，1314520 的二进制 10100000011011011000 中有 9 个 1，则 x=1314520 时程序输出 1。

```
var
     i,x,c:longint;
begin
     readln(x);
     c:=0;
     for i:=1 to 32 do
     begin
            c:=c + x and 1;
            x:=x shr 1;
     end;
     writeln( c and 1 );
```

```
end.
```

但这样的效率并不高，位运算的神奇之处还没有体现出来。

同样是判断二进制中 1 的个数的奇偶性，下面这段代码就强了。你能看出这个代码的原理吗？

```
var
     x:longint;
begin
     readln(x);
     x:=x xor (x shr 1);
     x:=x xor (x shr 2);
     x:=x xor (x shr 4);
     x:=x xor (x shr 8);
     x:=x xor (x shr 16);
     writeln(x and 1);
end.
```

为了说明上面这段代码的原理，我们还是拿 1314520 出来说事。1314520 的二进制为 10100000111011011000，第一次异或操作的结果如下：

```
         00000000000101000000111011011000
XOR      00000000000101000000011101101100
-----------------------------------------
         00000000000111100000100110110100
```

得到的结果是一个新的二进制数，其中右起第 i 位上的数表示原数中第 i 和 i+1 位上有奇数个 1 还是偶数个 1。比如，最右边那个 0 表示原数末两位有偶数个 1，右起第 3 位上的 1 就表示原数的这个位置和前一个位置中有奇数个 1。对这个数进行第二次异或的结果如下：

```
         00000000000111100000100110110100
XOR      00000000000111100000010011101101
-----------------------------------------
         00000000000110011000101111011001
```

结果里的每个 1 表示原数的该位置及其前面三个位置中共有奇数个 1，每个 0 就表示原数对应的四个位置上共偶数个 1。一直做到第五次异或结束后，得到的二进制数的最末位就表示整个 32 位数里有多少个 1，这就是我们最终想要的答案。

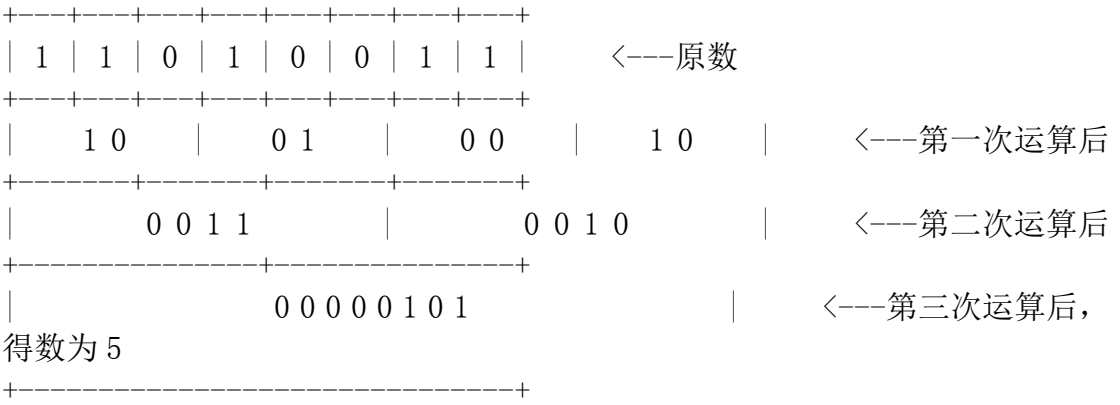
计算二进制中的 1 的个数

同样假设 x 是一个 32 位整数。经过下面五次赋值后，x 的值就是原数的二进制表示中数字 1 的个数。比如，初始时 x 为 1314520（网友抓狂：能不能换一个数啊），那么最后 x 就变成了 9，它表示 1314520 的二进制中有 9 个 1。

```
x := (x and $55555555) + ((x shr 1) and $55555555);
```

```
x := (x and $33333333) + ((x shr 2) and $33333333);
x := (x and $0F0F0F0F) + ((x shr 4) and $0F0F0F0F);
x := (x and $00FF00FF) + ((x shr 8) and $00FF00FF);
x := (x and $0000FFFF) + ((x shr 16) and $0000FFFF);
```

为了便于解说，我们下面仅说明这个程序是如何对一个 8 位整数进行处理的。我们拿数字 211（我们班某 MM 的生日）来开刀。211 的二进制为 11010011。



整个程序是一个分治的思想。第一次我们把每相邻的两位加起来，得到每两位里 1 的个数，比如前两位 10 就表示原数的前两位有 2 个 1。第二次我们继续两两相加，10+01=11，00+10=10，得到的结果是 00110010，它表示原数前 4 位有 3 个 1，末 4 位有 2 个 1。最后一次我们把 0011 和 0010 加起来，得到的就是整个二进制中 1 的个数。程序中巧妙地使用取位和右移，比如第二行中 $33333333 的二进制为 00110011001100....，用它和 x 做 and 运算就相当于以 2 为单位间隔取数。shr 的作用就是让加法运算的相同数位对齐。

二分查找 32 位整数的前导 0 个数

这里用的 C 语言，我直接 Copy 的 Hacker's Delight 上的代码。这段代码写成 C 要好看些，写成 Pascal 的话会出现很多 begin 和 end，搞得代码很难看。程序思想是二分查找，应该很简单，我就不细说了。

```
int nlz(unsigned x)
{
    int n;

    if (x == 0) return(32);
    n = 1;
    if ((x >> 16) == 0) {n = n +16; x = x <<16;}
    if ((x >> 24) == 0) {n = n + 8; x = x << 8;}
    if ((x >> 28) == 0) {n = n + 4; x = x << 4;}
    if ((x >> 30) == 0) {n = n + 2; x = x << 2;}
    n = n - (x >> 31);
    return n;
}
```

只用位运算来取绝对值

这是一个非常有趣的问题。大家先自己想想吧，Ctrl+A 显示答案。

答案：假设 x 为 32 位整数，则 x xor (not (x shr 31) + 1) + x shr 31 的结果是 x 的绝对值

x shr 31 是二进制的最高位，它用来表示 x 的符号。如果它为 0（x 为正），则 not (x shr 31) + 1 等于$00000000，异或任何数结果都不变；如果最高位为 1（x 为负），则 not (x shr 31) + 1 等于$FFFFFFFF，x 异或它相当于所有数位取反，异或完后再加一。

高低位交换

这个题实际上是我出的，做为学校内部 NOIp 模拟赛的第一题。题目是这样：

给出一个小于 2^32 的正整数。这个数可以用一个 32 位的二进制数表示（不足 32 位用 0 补足）。我们称这个二进制数的前 16 位为“高位”，后 16 位为“低位”。将它的高低位交换，我们可以得到一个新的数。试问这个新的数是多少（用十进制表示）。

例如，数 1314520 用二进制表示为 0000 0000 0001 0100 0000 1110 1101 1000（添加了 11 个前导 0 补足为 32 位），其中前 16 位为高位，即 0000 0000 0001 0100；后 16 位为低位，即 0000 1110 1101 1000。将它的高低位进行交换，我们得到了一个新的二进制数 0000 1110 1101 1000 0000 0000 0001 0100。它即是十进制的 249036820。

当时几乎没有人想到用一句位操作来代替冗长的程序。使用位运算的话两句话就完了。

```
var
     n:dword;
begin
     readln( n );
     writeln( (n shr 16) or (n     shl 16) );
end.
```

而事实上，Pascal 有一个系统函数 swap 直接就可以用。

二进制逆序

下面的程序读入一个 32 位整数并输出它的二进制倒序后所表示的数。

输入： 1314520 （二进制为 00000000000101000000111011011000）

输出： 460335104 （二进制为 00011011011100000010100000000000）

```
var
     x:dword;
```

```
begin
     readln(x);
     x := (x and $55555555) shl    1 or (x and $AAAAAAAA) shr    1;
     x := (x and $33333333) shl    2 or (x and $CCCCCCCC) shr    2;
     x := (x and $0F0F0F0F) shl    4 or (x and $F0F0F0F0) shr    4;
     x := (x and $00FF00FF) shl    8 or (x and $FF00FF00) shr    8;
     x := (x and $0000FFFF) shl 16 or (x and $FFFF0000) shr 16;
     writeln(x);
end.
```

它的原理和刚才求二进制中 1 的个数那个例题是大致相同的。程序首先交换每相邻两位上的数，以后把互相交换过的数看成一个整体，继续进行以 2 位为单位、以 4 位为单位的左右对换操作。我们再次用 8 位整数 211 来演示程序执行过程：

```
+---+---+---+---+---+---+---+---+
| 1 | 1 | 0 | 1 | 0 | 0 | 1 | 1 |     <---原数
+---+---+---+---+---+---+---+---+
|   1 1   |   1 0   |   0 0   |   1 1   |     <---第一次运算后
+-------+-------+-------+-------+
|       1 0 1 1       |       1 1 0 0       |     <---第二次运算后
+---------------+---------------+
|               1 1 0 0 1 0 1 1               |     <---第三次运算后
+-----------------------------------------------+
```

Copyright 也很强

```
writeln('Matrix', 42 XOR 105 , '原创，转贴请注明出处');
```

位运算简介及实用技巧（三）：进阶篇(2)

今天我们来看两个稍微复杂一点的例子。

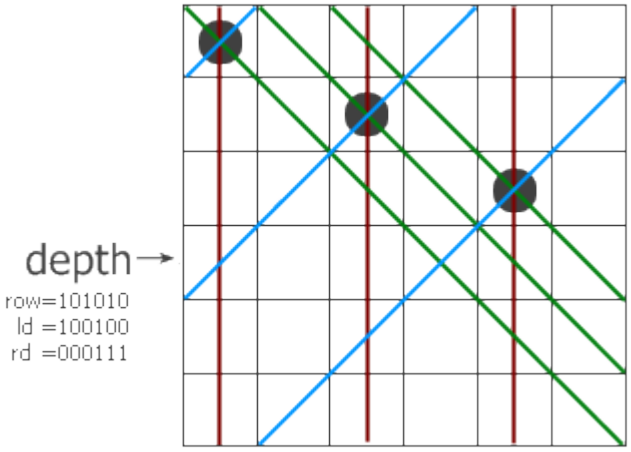
n 皇后问题位运算版

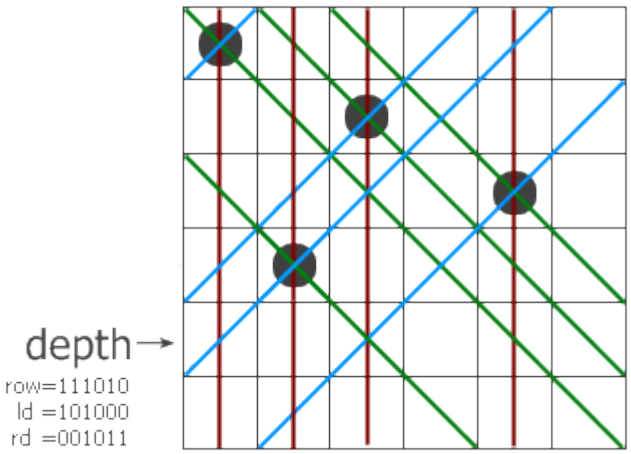
n 皇后问题是啥我就不说了吧，学编程的肯定都见过。下面的十多行代码是 n 皇后问题的一个高效位运算程序，看到过的人都夸它牛。初始时，upperlim:=(1 shl n)-1。主程序调用 test(0,0,0)后 sum 的值就是 n 皇后总的解数。拿这个去交 USACO，0.3s，暴爽。

```
procedure test(row,ld,rd:longint);
var
           pos,p:longint;
begin

{ 1}    if row<>upperlim then
{ 2}    begin
{ 3}         pos:=upperlim and not (row or ld or rd);
{ 4}         while pos<>0 do
{ 5}         begin
```

```
{ 6}                  p:=pos and -pos;
{ 7}                  pos:=pos-p;
{ 8}                  test(row+p,(ld+p)shl 1,(rd+p)shr 1);
{ 9}            end;
{10}      end
{11}      else inc(sum);

end;
```

乍一看似乎完全摸不着头脑，实际上整个程序是非常容易理解的。这里还是建议大家自己单步运行一探究竟，实在没研究出来再看下面的解说。





和普通算法一样，这是一个递归过程，程序一行一行地寻找可以放皇后的地方。过程带三个参数，row、ld 和 rd，分别表示在纵列和两个对角线方向的限制条件下这一行的哪些地方不能放。我们以 6x6 的棋盘为例，看看程序是怎么工作的。假设现在已经递归到第四层，前三层放的子已经标在左图上了。红色、蓝色和绿色的线分别表示三个方向上有冲突的位置，位于该行上的冲突位置就用 row、ld 和 rd 中的 1 来表示。把它们三个并起来，得到该行所有的禁位，取反后就得到所有可以放的位置（用 pos 来表示）。前面说过-a 相当于 not a + 1，这里的代码第 6 行就相当于 pos and (not pos + 1)，其结果是取出最右边的那个 1。这样，p 就表示该行的某个可以放子的位置，把它从 pos 中移除并递归调用 test 过程。注意递归调用时三个参数的变化，每个参数都加上了一个禁位，但两个对角线方向的禁位对下一行的影响

需要平移一位。最后，如果递归到某个时候发现 row=111111 了，说明六个皇后全放进去了，此时程序从第 1 行跳到第 11 行，找到的解的个数加一。

~~~~====~~~~=====　华丽的分割线　=====~~~~====~~~~

Gray 码

假如我有 4 个潜在的 GF，我需要决定最终到底和谁在一起。一个简单的办法就是，依次和每个 MM 交往一段时间，最后选择给我带来的“满意度”最大的 MM。但看了 dd 牛的理论后，事情开始变得复杂了：我可以选择和多个 MM 在一起。这样，需要考核的状态变成了 2^4=16 种（当然包括 0000 这一状态，因为我有可能是玻璃）。现在的问题就是，我应该用什么顺序来遍历这 16 种状态呢？

传统的做法是，用二进制数的顺序来遍历所有可能的组合。也就是说，我需要以 0000->0001->0010->0011->0100->...->1111 这样的顺序对每种状态进行测试。这个顺序很不科学，很多时候状态的转移都很耗时。比如从 0111 到 1000 时我需要暂时甩掉当前所有的 3 个 MM，然后去把第 4 个 MM。同时改变所有 MM 与我的关系是一件何等巨大的工程啊。因此，我希望知道，是否有一种方法可以使得，从没有 MM 这一状态出发，每次只改变我和一个 MM 的关系（追或者甩），15 次操作后恰好遍历完所有可能的组合（最终状态不一定是 1111）。大家自己先试一试看行不行。

解决这个问题的方法很巧妙。我们来说明，假如我们已经知道了 n=2 时的合法遍历顺序，我们如何得到 n=3 的遍历顺序。显然，n=2 的遍历顺序如下：

00
01
11
10

你可能已经想到了如何把上面的遍历顺序扩展到 n=3 的情况。n=3 时一共有 8 种状态，其中前面 4 个把 n=2 的遍历顺序照搬下来，然后把它们对称翻折下去并在最前面加上 1 作为后面 4 个状态：

000
001
011
010 ↑
----
110 ↓
111
101
100

用这种方法得到的遍历顺序显然符合要求。首先，上面 8 个状态恰好是 n=3 时的所有 8 种组合，因为它们是在 n=2 的全部四种组合的基础上考虑选不选第 3 个元素所得到的。然后我们看到，后面一半的状态应该和前面一半一样满足“相邻状态间仅一位不同”的限制，而“镜面”处则是最前面那一位数不同。再次翻折三阶遍历顺序，我们就得到了刚才的问题的答案：
~~~~

0000
0001
0011
0010
0110
0111
0101
0100
1100
1101
1111
1110
1010
1011
1001
1000

这种遍历顺序作为一种编码方式存在，叫做 Gray 码（写个中文让蜘蛛来抓：格雷码）。它的应用范围很广。比如，n 阶的 Gray 码相当于在 n 维立方体上的 Hamilton 回路，因为沿着立方体上的边走一步，n 维坐标中只会有一个值改变。再比如，Gray 码和 Hanoi 塔问题等价。Gray 码改变的是第几个数，Hanoi 塔就该移动哪个盘子。比如，3 阶的 Gray 码每次改变的元素所在位置依次为 1-2-1-3-1-2-1，这正好是 3 阶 Hanoi 塔每次移动盘子编号。如果我们可以快速求出 Gray 码的第 n 个数是多少，我们就可以输出任意步数后 Hanoi 塔的移动步骤。现在我告诉你，Gray 码的第 n 个数（从 0 算起）是 n xor (n shr 1)，你能想出来这是为什么吗？先自己想想吧。

下面我们把二进制数和 Gray 码都写在下面，可以看到左边的数异或自身右移的结果就等于右边的数。

二进制数	Gray 码
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

从二进制数的角度看，“镜像”位置上的数即是对原数进行 not 运算后的结果。比如，第 3 个数 010 和倒数第 3 个数 101 的每一位都正好相反。假设这两个数分别为 x 和 y，那么 x xor (x shr 1)和 y xor (y shr 1)的结果只有一点不同：后者的首位是 1，前者的首位是 0。而这正好是 Gray 码的生成方法。这就说明了，Gray 码的第 n 个数确实是 n xor (n shr 1)。

今年四月份 mashuo 给我看了这道题，是二维意义上的 Gray 码。题目大意是说，把 0 到 2^(n+m)-1 的数写成 2^n * 2^m 的矩阵，使得位置相邻两数的二进制表示只有一位之差。答案其实很简单，所有数都是由 m 位的 Gray 码和 n 位 Gray 码拼接而成，需要用左移操作和 or 运算完成。完整的代码如下：

```
var
     x,y,m,n,u:longint;
begin
     readln(m,n);
     for x:=0 to 1 shl m-1 do begin
           u:=(x xor (x shr 1)) shl n; //输出数的左边是一个 m 位的 Gray 码
           for y:=0 to 1 shl n-1 do
                write(u or (y xor (y shr 1)),' '); //并上一个 n 位 Gray 码
           writeln;
     end;
end.
```

Matrix67 原创
转贴请注明出处
位运算简介及实用技巧（四）：实战篇

下面分享的是我自己写的三个代码，里面有些题目也是我自己出的。这些代码都是在我的 Pascal 时代写的，恕不提供 C 语言了。代码写得并不好，我只是想告诉大家位运算在实战中的应用，包括了搜索和状态压缩 DP 方面的题目。其实大家可以在网上找到更多用位运算优化的题目，这里整理出一些自己写的代码，只是为了原创系列文章的完整性。这一系列文章到这里就结束了，希望大家能有所收获。

Matrix67 原创，转贴请注明出处。

Problem : 费解的开关

题目来源

06 年 NOIp 模拟赛（一） by Matrix67 第四题

问题描述

你玩过“拉灯”游戏吗？25 盏灯排成一个 5x5 的方形。每一个灯都有一个开关，游戏者可以改变它的状态。每一步，游戏者可以改变某一个灯的状态。游戏者改变一个灯的状态会产生连锁反应：和这个灯上下左右相邻的灯也要相应地改变其状态。

我们用数字“1”表示一盏开着的灯，用数字“0”表示关着的灯。下面这种状态

10111
01101
10111
10000
11011

在改变了最左上角的灯的状态后将变成：

01111
11101
10111
10000
11011

再改变它正中间的灯后状态将变成：

01111
11001
11001
10100
11011

给定一些游戏的初始状态，编写程序判断游戏者是否可能在 6 步以内使所有的灯都变亮。

输入格式

第一行有一个正整数 n，代表数据中共有 n 个待解决的游戏初始状态。

以下若干行数据分为 n 组，每组数据有 5 行，每行 5 个字符。每组数据描述了一个游戏的初始状态。各组数据间用一个空行分隔。

对于 30%的数据，n<=5；

对于 100%的数据，n<=500。

输出格式

输出数据一共有 n 行，每行有一个小于等于 6 的整数，它表示对于输入数据中对应的游戏状态最少需要几步才能使所有灯变亮。

对于某一个游戏初始状态，若 6 步以内无法使所有灯变亮，请输出“-1”。

样例输入

3
00111
01011
10001
11010

11100

11101
11101
11110
11111
11111

01111
11111
11111
11111
11111

样例输出
3
2
-1

程序代码

```
const
     BigPrime=3214567;
     MaxStep=6;
type
     pointer=^rec;
     rec=record
                 v:longint;
                 step:integer;
                 next:pointer;
             end;

var
     total:longint;
     hash:array[0..BigPrime-1]of pointer;
     q:array[1..400000]of rec;

function update(a:longint;p:integer):longint;
begin
     a:=a xor (1 shl p);
     if p mod 5<>0 then a:=a xor (1 shl (p-1));
     if (p+1) mod 5<>0 then a:=a xor (1 shl (p+1));
     if p<20 then a:=a xor (1 shl (p+5));
```

```
     if p>4 then a:=a xor (1 shl (p-5));
     exit(a);
end;

function find(a:longint;step:integer):boolean;
var
     now:pointer;
begin
     now:=hash[a mod BigPrime];
     while now<>nil do
     begin
            if now^.v=a then exit(true);
            now:=now^.next;
     end;

     new(now);
     now^.v:=a;
     now^.step:=step;
     now^.next:=hash[a mod BigPrime];
     hash[a mod BigPrime]:=now;
     total:=total+1;
     exit(false);
end;

procedure solve;
var
     p:integer;
     close:longint=0;
     open:longint=1;
begin
     find(1 shl 25-1,0);
     q[1].v:=1 shl 25-1;
     q[1].step:=0;
     repeat
            inc(close);
            for p:=0 to 24 do
                 if not find(update(q[close].v,p),q[close].step+1) and
(q[close].step+1<MaxStep) then
                 begin
                        open:=open+1;
                        q[open].v:=update(q[close].v,p);
                        q[open].step:=q[close].step+1;
                 end;
     until close>=open;
```

```
end;

procedure print(a:longint);
var
     now:pointer;
begin
     now:=hash[a mod BigPrime];
     while now<>nil do
     begin
            if now^.v=a then
            begin
                 writeln(now^.step);
                 exit;
            end;
            now:=now^.next;
     end;
     writeln(-1);
end;

procedure main;
var
     ch:char;
     i,j,n:integer;
     t:longint;
begin
     readln(n);
     for i:=1 to n do
     begin
            t:=0;
            for j:=1 to 25 do
            begin
                 read(ch);
                 t:=t*2+ord(ch)-48;
                 if j mod 5=0 then readln;
            end;
            print(t);
            if i<n then readln;
     end;
end;

begin
     solve;
     main;
end.
```

========== **性感的分割线** ==========

Problem : garden / 和 MM 逛花园

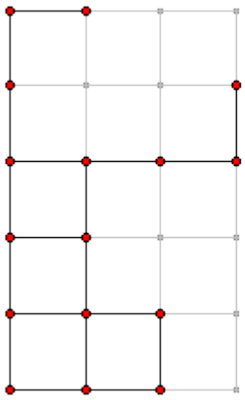
题目来源

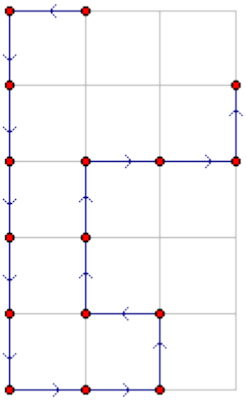
07 年 Matrix67 生日邀请赛第四题

问题描述

花园设计强调，简单就是美。Matrix67 常去的花园有着非常简单的布局：花园的所有景点的位置都是“对齐”了的，这些景点可以看作是平面坐标上的格点。相邻的景点之间有小路相连，这些小路全部平行于坐标轴。景点和小路组成了一个“不完整的网格”。

一个典型的花园布局如左图所示。花园布局在 6 行 4 列的网格上，花园的 16 个景点的位置用红色标注在了图中。黑色线条表示景点间的小路，其余灰色部分实际并不存在。





Matrix67 的生日那天，他要带着他的 MM 在花园里游玩。Matrix67 不会带 MM 两次经过同一个景点，因此每个景点最多被游览一次。他和他的 MM 边走边聊，他们是如此的投入以致于他们从不会“主动地拐弯”。也就是说，除非前方已没有景点或是前方的景点已经访问过，否则他们会一直往前走下去。当前方景点不存在或已游览过时，Matrix67 会带 MM 另选一个方向继续前进。由于景点个数有限，访问过的景点将越来越多，迟早会出现不能再走的情况（即四个方向上的相邻景点都访问过了），此时他们将结束花园的游览。Matrix67 希望知道以这种方式游览花园是否有可能遍历所有的景点。Matrix67 可以选择从任意一个景点开始游览，以任意一个景点结束。

在上图所示的花园布局中，一种可能的游览方式如右图所示。这种浏览方式从(1,2)出发，以(2,4)结束，经过每个景点恰好一次。

输入格式

第一行输入两个用空格隔开的正整数 m 和 n，表示花园被布局在 m 行 n 列的网格上。

以下 m 行每行 n 个字符，字符“0”表示该位置没有景点，字符“1”表示对应位置有景点。这些数字之间没有空格。

输出格式

你的程序需要寻找满足“不主动拐弯”性质且遍历所有景点的游览路线。

如果没有这样的游览路线，请输出一行“Impossible”（不带引号，注意大小写）。

如果存在游览路线，请依次输出你的方案中访问的景点的坐标，每行输出一个。坐标的表示格式为“(x,y)”，代表第 x 行第 y 列。

如果有多种方案，你只需要输出其中一种即可。评测系统可以判断你的方案的正确性。

样例输入

```
6 4
1100
1001
1111
1100
1110
1110
```

样例输出

```
(1,2)
(1,1)
(2,1)
(3,1)
(4,1)
(5,1)
(6,1)
(6,2)
(6,3)
(5,3)
(5,2)
(4,2)
(3,2)
(3,3)
(3,4)
(2,4)
```

数据规模

对于 30%的数据，n,m<=5；

对于 100%的数据，n,m<=10。

程序代码：

```
program garden;

const
     dir:array[1..4,1..2]of integer=
         ((1,0),(0,1),(-1,0),(0,-1));

type
     arr=array[1..10]of integer;
     rec=record x,y:integer;end;

var
     map:array[0..11,0..11]of boolean;
     ans:array[1..100]of rec;
     n,m,max:integer;
     step:integer=1;
     state:arr;

procedure readp;
var
     i,j:integer;
     ch:char;
begin
     readln(m,n);
     for i:=1 to n do
     begin
            for j:=1 to m do
            begin
                 read(ch);
                 map[i,j]:=(ch='1');
                 inc(max,ord( map[i,j] ))
            end;
     readln;
     end;
end;

procedure writep;
var
     i:integer;
begin
     for i:=1 to step do
            writeln( '(' , ans[i].x , ',' , ans[i].y , ')' );
```

```
end;

procedure solve(x,y:integer);
var
     tx,ty,d:integer;
     step_cache:integer;
     state_cache:arr;
begin
     step_cache:=step;
     state_cache:=state;
     if step=max then
     begin
            writep;
            exit;
     end;

     for d:=1 to 4 do
     begin
            tx:=x+dir[d,1];
            ty:=y+dir[d,2];
            while map[tx,ty] and ( not state[tx] and(1 shl (ty-1) )>0)
do
            begin
                 inc(step);
                 ans[step].x:=tx;
                 ans[step].y:=ty;
                 state[tx]:=state[tx] or ( 1 shl (ty-1) );
                 tx:=tx+dir[d,1];
                 ty:=ty+dir[d,2];
            end;

            tx:=tx-dir[d,1];
            ty:=ty-dir[d,2];
            if (tx<>x) or (ty<>y) then solve(tx,ty);
            state:=state_cache;
            step:=step_cache;
     end;
end;

{====main====}
var
     i,j:integer;
begin
     assign(input,'garden.in');
```

```
    reset(input);
    assign(output,'garden.out');
    rewrite(output);

    readp;
    for i:=1 to n do
    for j:=1 to m do
        if map[i,j] then
        begin
               ans[1].x:=i;
               ans[1].y:=j;
               state[i]:=1 shl (j-1);
               solve(i,j);
               state[i]:=0;
        end;

    close(input);
    close(output);
end.
```

========== **性感的分割线** ==========

Problem : cowfood / 玉米地

题目来源

USACO月赛

问题描述

农夫约翰购买了一处肥沃的矩形牧场，分成M*N(1<=M<=12; 1<=N<=12)个格子。他想在那里的一些格子中种植美味的玉米。遗憾的是，有些格子区域的土地是贫瘠的，不能耕种。

精明的约翰知道奶牛们进食时不喜欢和别的牛相邻，所以一旦在一个格子中种植玉米，那么他就不会在相邻的格子中种植，即没有两个被选中的格子拥有公共边。他还没有最终确定哪些格子要选择种植玉米。

作为一个思想开明的人，农夫约翰希望考虑所有可行的选择格子种植方案。由于太开明，他还考虑一个格子都不选择的种植方案！请帮助农夫约翰确定种植方案总数。

输入格式:

第一行：两个用空格分隔的整数M和N

第二行到第M+1行：第i+1行描述牧场第i行每个格子的情况，N个用空格分隔的整数，表示这个格子是否可以种植（1表示肥沃的、适合种植，0表示贫瘠的、不可种植）

输出格式

一个整数，农夫约翰可选择的方案总数除以 100,000,000 的余数

样例输入

2 3
1 1 1
0 1 0

样例输出

9

样例说明

给可以种植玉米的格子编号：

1 2 3
4

只种一个格子的方案有四种(1,2,3 或 4)，种植两个格子的方案有三种(13,14 或 34)，种植三个格子的方案有一种(134)，还有一种什么格子都不种。

4+3+1+1=9。

数据规模

对于 30%的数据，N,M<=4；

对于 100%的数据，N,M<=12。

程序代码：

```
program cowfood;

const
     d=100000000;
     MaxN=12;

var
     f:array[0..MaxN,1..2000]of longint;
     w:array[1..2000,1..2000]of boolean;
     st:array[0..2000]of integer;
     map:array[0..MaxN]of integer;
     m,n:integer;

function Impossible(a:integer):boolean;
var
```

```
     i:integer;
     flag:boolean=false;
begin
     for i:=1 to MaxN do
     begin
            if flag and (a and 1=1) then exit(true);
            flag:=(a and 1=1);
            a:=a shr 1;
     end;
     exit(false);
end;

function Conflict(a,b:integer):boolean;
var
     i:integer;
begin
     for i:=1 to MaxN do
     begin
            if (a and 1=1) and (b and 1=1) then exit(true);
            a:=a shr 1;
            b:=b shr 1;
     end;
     exit(false);
end;

function CanPlace(a,b:integer):boolean;
begin
     exit(a or b=b);
end;

procedure FindSt;
var
     i:integer;
begin
     for i:=0 to 1 shl MaxN-1 do
            if not Impossible(i) then
            begin
                 inc(st[0]);
                 st[st[0]]:=i;
            end;
end;

procedure Init;
var
```

```
     i,j:integer;
begin
     for i:=1 to st[0] do
     for j:=i to st[0] do
            if not Conflict(st[i],st[j]) then
            begin
                 w[i,j]:=true;
                 w[j,i]:=true;
            end;
end;

procedure Readp;
var
     i,j,t,v:integer;
begin
     readln(m,n);
     for i:=1 to m do
     begin
            v:=0;
            for j:=1 to n do
            begin
                 read(t);
                 v:=v*2+t;
            end;
            map[i]:=v;
            readln;
     end;
end;

procedure Solve;
var
     i,j,k:integer;
begin
     f[0,1]:=1;
     map[0]:=1 shl n-1;
     for i:=1 to m do
     for j:=1 to st[0] do
            if not CanPlace(st[j],map[i]) then f[i,j]:=-1 else
                for k:=1 to st[0] do if (f[i-1,k]<>-1) and w[j,k] then
                     f[i,j]:=(f[i,j]+f[i-1,k]) mod d;
end;

procedure Writep;
var
```

```
     j:integer;
     ans:longint=0;
begin
     for j:=1 to st[0] do
            if f[m,j]<>-1 then ans:=(ans+f[m,j]) mod d;
     writeln(ans);
end;

begin
     assign(input,'cowfood.in');
     reset(input);
     assign(output,'cowfood.out');
     rewrite(output);

     FindSt;
     Init;
     Readp;
     Solve;
     Writep;

     close(input);
     close(output);
end.
```