位运算

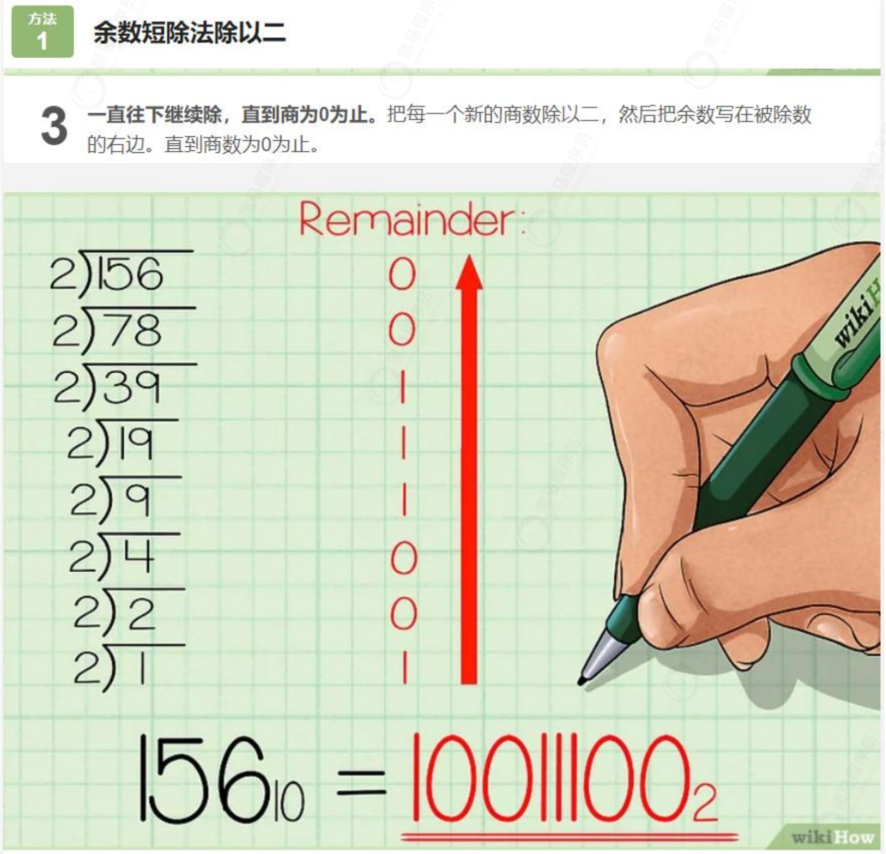
讲师：唐僧

多年一线研发经验，项目覆盖电商，金融，办公自动化，在线教育等，有着非常丰富的项目开发经验，且致力于研究大厂面试算法多年，有着丰富的算法面经，另外在大数据，物联网等方面也有着深入的研究。

为什么讲位运算?

计算机中的数在内存中都是以二进制形式进行存储的，用位运算就是直接对整数在内存中的二进制位进行操作，因此其执行效率非常高，在程序中尽量使用位运算进行操作，这会大大提高程序的性能。

十进制 <——> 二进制



10进制	2进制
4	0100
5	0101
6	0110
8	01000

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	0	1	0	1	0	1	0

在线进制转换：https://tool.oschina.net/hexconvert/

链接：https://zh.wikihow.com/从十进制转换为二进制
（均以整数为例说明，后同）

常见的位运算符

运算符	作用	示例	说明
&	**与运算** 两个位都是 1 时，结果才为 1，否则为 0	1 0 0 1 1 & 1 1 0 0 1 ------------------------------ 1 0 0 0 1	
|	**或运算** 两个位都是 0 时，结果才为 0，否则为 1	1 0 0 1 1 | 1 1 0 0 1 ------------------------------ 1 1 0 1 1	
^	**异或运算**， 两个位相同则为 0，不同则为 1	1 0 0 1 1 ^ 1 1 0 0 1 ----------------------------- 0 1 0 1 0	也可用不进位加法来理解
~	**取反运算**， 0 则变为 1，1 则变为 0	~ 1 0 0 1 1 ----------------------------- 0 1 1 0 0	

常见的位运算符

运算符	作用	示例	说明
<<	**左移运算**， 向左进行移位操作，高位丢弃，低位补 0	bit=0000 1000 bit << 3 移位前：0000 1000 移位后：0100 0000	以8位二进制说明
>>	**右移运算**， 向右进行移位操作，对**无符号数，高位补 0**，对于**有符号数，高位补符号位**	bit=0000 1000 bit >>3 移位前：0000 1000 移位后：0000 0001 bit=1111 1000 bit >>3 移位前：1111 1000 移位后：1111 1111	以8位二进制说明

有符号数&无符号数

场景：假设用1字节8位二进制表示一个数，那么8可以表示为：0000 1000，请问-8应该如何表示?

日常书写表示：负数其实就是正数前面加了一个负号

计算机内部表示：将二进制数的最高位（第一位）规定为符号位，用0代表正数，1代表负数，就能够表示负数了
这样八位二进制数中就分成了第一位符号位和后七位数值位

结论：有符号数中最高位用于表示正负，无符号数中，所有的位都用于直接表示该值的大小

不同字节下有符号和无符号数能表示的范围如下：

	1字节	2字节	4字节
无符号数	0~255	0~65535	0~4294967295
有符号数	-128~127	-32768~32765	-2147483648~2147483647

注意：java没有无符号类型，都是有符号类型的数据类型

计算机如何存储负数

请问：-2是否可表示为：1000 0010

请计算：1 +（-2）

0000 0001
1000 0010

1000 0011

结果为：-3，显然不对

符号位也一起代入运算了，而符号位是额外人为规定的，本身并不符合算术运算的规则。因此还要为符号位单独制定一套运算规则

反码：正数不变，负数的每一数值位都取反（首位符号保持不变），用反码，符号位也可以直接代入运算了

0000 0001
1111 1101

1111 1110

首位1，为负数，负数用反码表示
则**原码**为：1000 0001，答案为-1，正确

那 1-1呢?

0000 0001
1111 1110

1111 1111

取反得到原码10000000，也就是-0。-0就是0，但这样一来，0这个数就有了两种表示方式。日常书写时不会写-0，但计算机内部的0和-0却是真实同时存在的。同时因为0的重复记录，八位二进制共256个位置却只表示了-127到127一共255个数，浪费了一个位置。

计算机如何存储负数

知识小贴士：已知一个数的补码，求原码的操作分两种情况：
（1）如果补码的符号位为“0”，表示是一个正数，所以补码就是该数的原码。
（2）如果补码的符号位为“1”，表示是一个负数，求原码的操作可以是：符号位为1，其余各位取反，然后再整个数加1。

负数用补码表示：补码=反码+1

以-1为例，原码10000001，反码11111110，补码11111111。

请计算：1+（-1）

```
      0000 0001
      1111 1111
------------------------------
    1 0000 0000
```

由于只有八位二进制数，进位产生的最高位1会被舍弃，最终结果就是0000 0000，也就是0，现在-0消失了

空出来的位置呢？尝试计算：-127-1=？，
10000001+11111111=（1）1000 0000。因为补码=反码+1，发现1000 0000这个补码不存在对应的反码，自然也没有原码，这个就是-0被去掉后多出的位置。但根据“-127-1=-128”的等式，把10000000规定为-128的补码

结论：按照补码的方案，八位二进制有符号数的表示范围就是-128到127共256个数。n位二进制有符号数的表示范围是-2^（n-1）到2^（n-1）-1共2^n个数

常见的位运算符

运算符	作用	示例	说明
>>	有符号右移， 无符号数，高位补 0， 有符号数，高位补符号位	bit=0000 1000 bit >>3 移位前：0000 1000 移位后：0000 0001 bit=1111 1000 bit >>3 移位前：1111 1000 移位后：1111 1111	以8位有符号二进制说明
>>>	无符号右移 右移后空位不管你符号位是啥，我都只填0。	bit=1111 1000 bit >>>3 移位前：1111 1000 移位后：0001 1111	

常见位运算奇技淫巧

常见位运算问题

以1字节8位二进制整数为例（后同）

位操作实现乘除法

数 a 向右移一位，相当于将 a 除以 2；
数 a 向左移一位，相当于将 a 乘以 2

2<<1 =4	2>>1 =1
0000 0010	0000 0010
0000 0100	0000 0001

神奇的异或操作

x ^ 0 = x	x ^ x = 0	x ^ 1s = ~x 注：1s = ~0	x ^ (~x) = 1s
0010 ^ 0000 ---------- 0010	0010 ^ 0010 ---------- 0000	0010 ^ 1111 ---------- 1101	0010 ^ 1101 ---------- 1111

位操作交换两数

位操作交换两数可以不需要第三个临时变量，虽然普通操作也可以做到，但是没有其效率高

```
//普通操作
void swap(int a, int b) {
  a = a + b;
  b = a - b;
  a = a - b;
}
```

```
//位操作
void swap(int a, int b) {
  a ^= b;
  b ^= a;
  a ^= b;
}
```

第一步：a ^= b ---> a = (a^b);
第二步：b ^= a ---> b = b^(a^b) ---> b = (b^b)^a = a
第三步：a ^= b ---> a = (a^b)^a = (a^a)^b = b

常见位运算问题

位操作判断奇偶数

只要根据数的最后一位是 0 还是 1 来决定即可，为 0 就是偶数，为 1 就是奇数

```
//普通操作
if ( a % 2 ==0) {
    //偶数
}else {

}
```

```
//位操作
if ( (a & 1) ==0 ) {
    //偶数
}else {

}
```

位操作求绝对值

整数的绝对值是其本身，负数的绝对值正好可以对其进行取反加一求得

```
int abs(int a) {
  int i = a >> 31;
  return i == 0 ? a : (~a + 1);
}
```

```
//优化后
int abs(int a) {
  int i = a >> 31;
  return ((a^i) - i);
}
```

位操作进行高低位交换

给定一个 16 位的无符号整数，将其高 8 位与低 8 位进行交换，求出交换后的值，如：

34520的二进制表示：
10000110 11011000
将其高8位与低8位进行交换，得到一个新的二进制数：
11011000 10000110
其十进制为55430

只要将无符号数 a>>8 即可得到其高 8 位移到低 8 位，高位补 0；
将 a<<8 即可将 低 8 位移到高 8 位，低 8 位补 0，
然后将 a>>8 和 a<<8 进行或操作既可求得交换后的结果。

```
a = (a >> 8) | (a << 8);
```

指定位置的位运算

注意：这里第n位是从0开始算

1、将x最右边的n位清零：x & ((~0) << n)

2、获取x的第n位的值(0或1)： （x>>n）& 1

3、获取x的第n位的幂值： x & （1<<n）

4、将x的第n位置为1： x | （1 << n ）

5、将x的第n位置为0： x & （~（1<<n））

6、将x的最高位至第n位（含第n位）清零 x & （（1<<n）-1）

7、将x的二进制表示中的最低位的1置成0：x & (x−1)

```
  0010 0110
& 0010 0101
  0010 0100
```

即：消去最低位的1

8、获取x的二进制表示中的最低位的1的位置：x & (−x)

```
  0010 0110
& 1101 1010
  0000 0010
```

9、x & ~x = 0

面试题集锦

191. 位1的个数

191. 位1的个数

难度 简单 243

编写一个函数，输入是一个无符号整数（以二进制串的形式），返回其二进制表达式中数字位数为 '1' 的个数（也被称为汉明重量）。

提示：

- 请注意，在某些语言（如 Java）中，没有无符号整数类型。在这种情况下，输入和输出都将被指定为有符号整数类型，并且不应影响您的实现，因为无论整数是有符号的还是无符号的，其内部的二进制表示形式都是相同的。
- 在 Java 中，编译器使用二进制补码记法来表示有符号整数。因此，在上面的 **示例 3** 中，输入表示有符号整数 -3。

进阶：

- 如果多次调用这个函数，你将如何优化你的算法？

解法一：位移+计数

n & 1 ==1：最低位是1，计数器+1

n>>1，继续判断

移位32次结束（java中int用4字节表示）

解法二：n & (n－1)消去最低位的1

n & (n－1)，可消去n的二进制表示中最后一位1，消去1次，计数器+1

当n==0时，所有的1都被置为了0，结束

231. 2的幂

231. 2的幂

难度 简单 268

给定一个整数，编写一个函数来判断它是否是 2 的幂次方。

示例 1:

```
输入：1
输出：true
解释：2^0 = 1
```

示例 2:

```
输入：16
输出：true
解释：2^4 = 16
```

知识点：如果一个数是2的幂次方则只会有一个二进制位为1

比如：int a = 32，其二进制表示为

00000000 00000000 00000000 00100000

- **解决方案：n & (n-1) == 0，则证明该数是2的幂次方**

190. 颠倒二进制位

190. 颠倒二进制位

难度 简单 250

颠倒给定的 32 位无符号整数的二进制位。

示例 1：

```
输入：00000010100101000001111010011100
输出：00111001011110000010100101000000
解释：输入的二进制串 00000010100101000001111010011100 表示无符号整数 43261596，
      因此返回 964176192，其二进制表示形式为 00111001011110000010100101000000。
```

知识小贴士：异或操作可以看作不进位加法

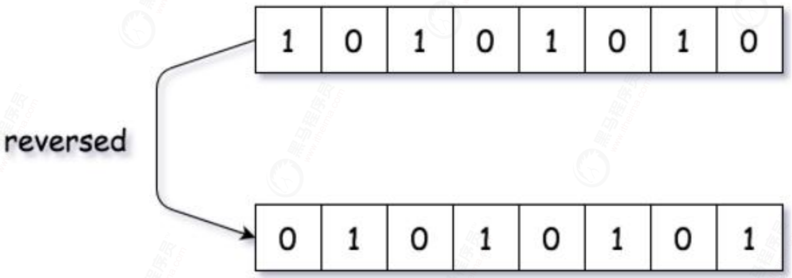
解法一：取模求和

十进制： `ans = ans * 10 + n % 10; n = n / 10;`

二进制： `ans = ans * 2 + n % 2; n = n / 2;`

```java
public int reverseBits(int n) {
    int res = 0;
    for (int i = 0; i < 32; i++) {
        res = (res << 1) + (n & 1);
        n >>= 1;
    }
    return res;
}
```

解法二：按位翻转



对于位于索引 i 处的位，在反转之后其位置应为 31-i（注：索引从零开始）

遍历32次，i从0到31，i=0代表最低位得到第i位的值，即翻转后第31-i 位的值

1 & (n >> i) == 0表明第i位是0，否则是1

定义结果res，从高位依次填充值

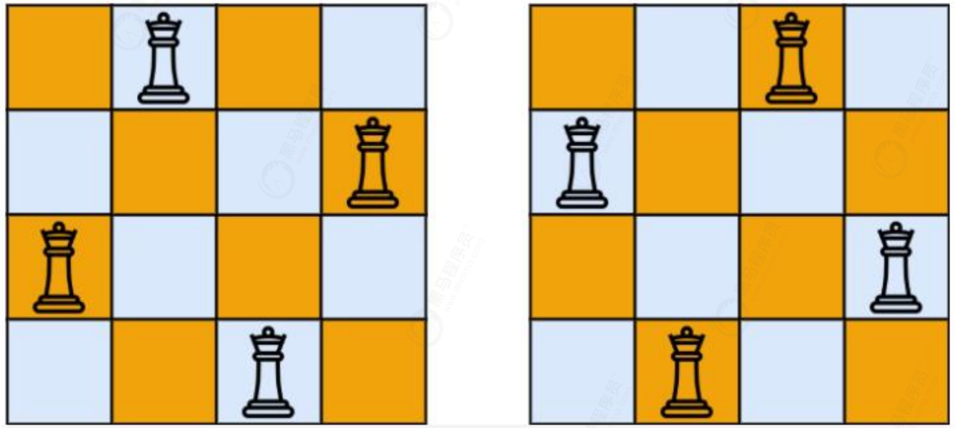
如果是1： res ^= 1<<(31-i)
如果是0： res ^= 0

52. N皇后 II

n 皇后问题 研究的是如何将 `n` 个皇后放置在 `n×n` 的棋盘上，并且使皇后彼此之间不能相互攻击。

给你一个整数 `n` ，返回 **n 皇后问题** 不同的解决方案的数量。

示例 1：



```
输入：n = 4
输出：2
解释：如上图所示，4 皇后问题存在两个不同的解法。
```

解法1：基于集合的回溯

使用三个Set分别代表列，撇，捺，三个方向上是否已有皇后

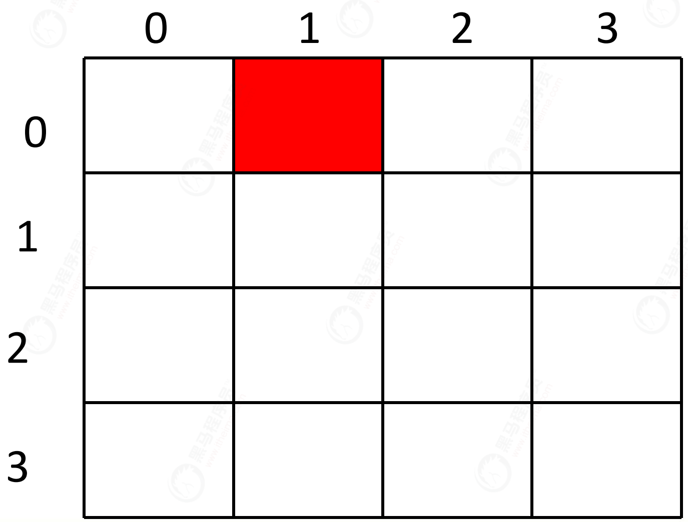
存储皇后信息的空间复杂度为O(n)

解法2：基于位运算的回溯

用三个整数col,pie,na分别记录在每行选择位置时列，撇，捺三个方向是否有皇后

利用整数的二进制位来代表棋盘每列是否有皇后，对于N皇后我们需要使用整数中N个二进制位

（java中的int最多可以表示32皇后，以4皇后为例说明）



0：该列无皇后，可以放置

1：该列有皇后，不可放置

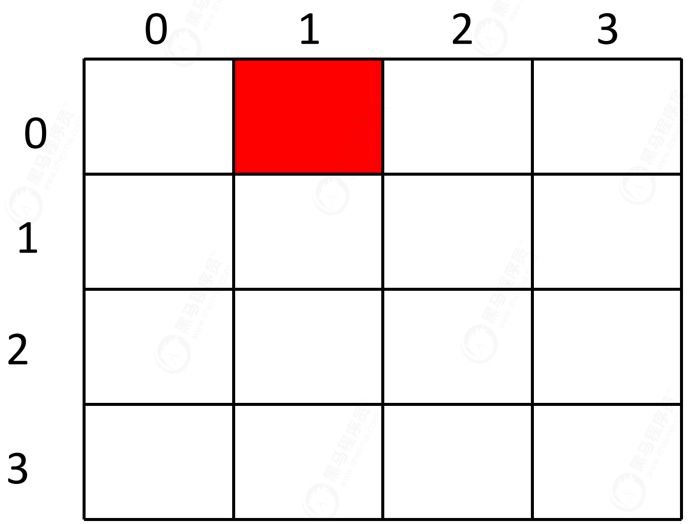
col=0010

pie=0010　撇和捺转换为列

na= 0010

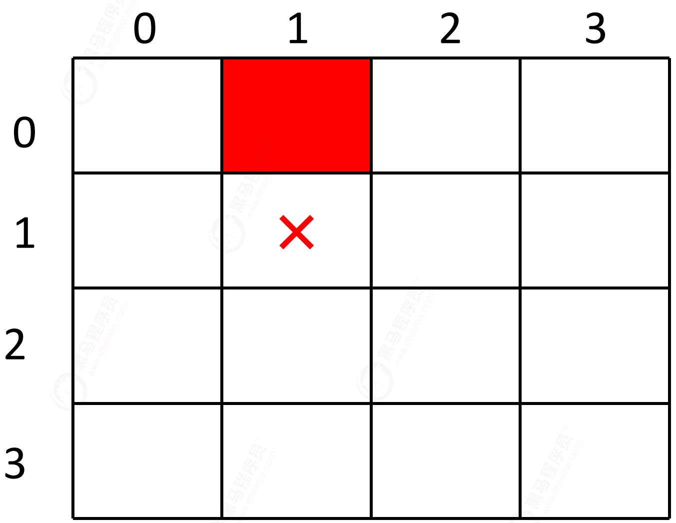
最左列对应最低二进制位，最右列对应最高二进制位

52. N皇后 II-如何更新二进制数



如图：已分别在棋盘第0行的第1列放置了皇后(下标从0开始)，现在要在第1行放置皇后，哪些位置不能放呢？

不同列
col=0010



col=0010，pie=0001，na= 0100，
就是在当前行放皇后的所有限制条件

故：第1行能放的位置只有第3列

将第3列放置皇后后二进制数如何变?

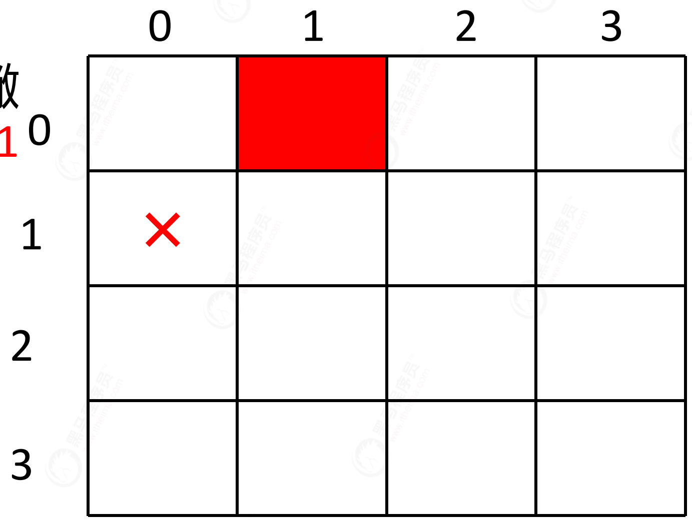
col=0010
pie=0001
na= 0100

| 1000 →

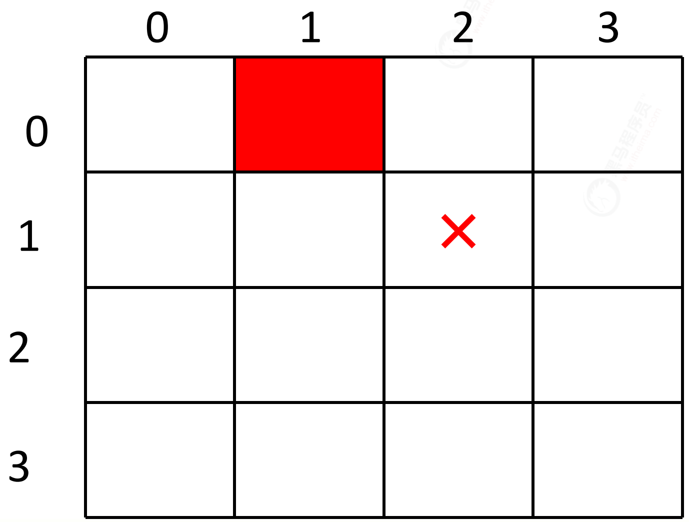
col=1010
pie=1001
na= 1100

（或运算，在第3列放皇后表示为1000）

不同撇
pie=0001



总结：在某一行如何做选择?
如果放在第i列，则将三个数第i个二进制位置1，通过或运算解决(从低到高的第i个二进制位)



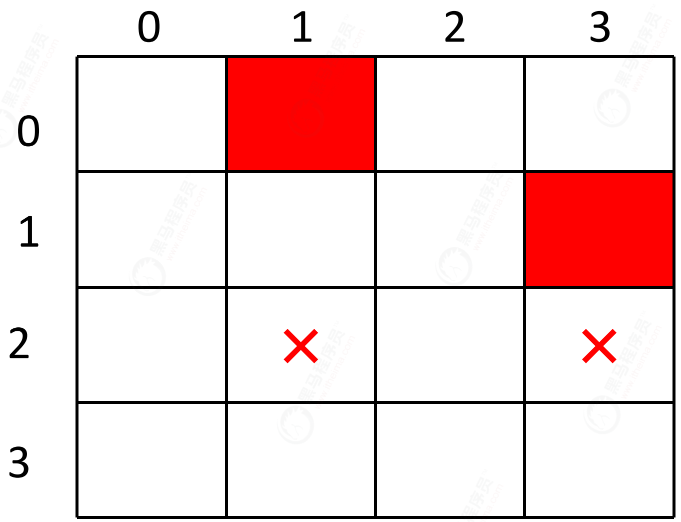
不同捺
na=0100

（新问题：每行的三个限制如何得来？）

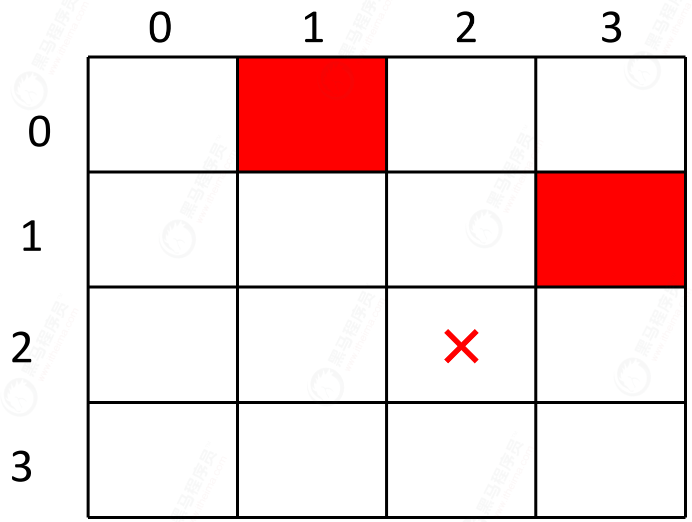
52. N皇后 II-如何更新二进制数

col=1010
pie=1001
na= 1100

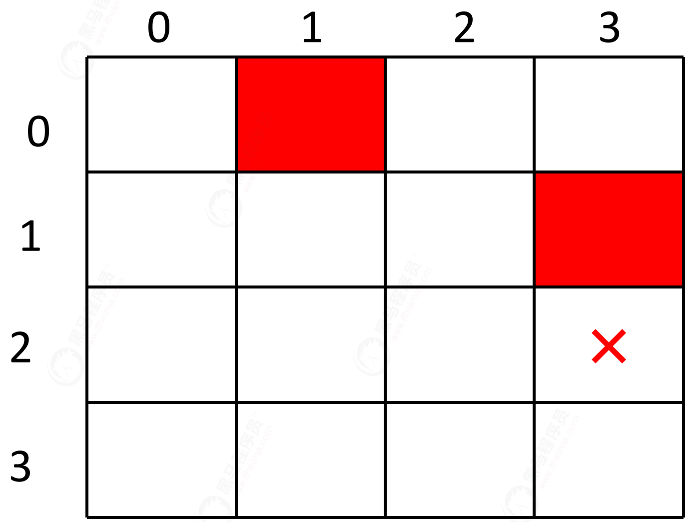
接着进行第2行，哪些位置不能放呢? col,pie,na作为当前行的判重条件是如何从上一行的结果转换而来的?



不同列



不同撇



不同捺

二进制数如何变化:

col=1010

col=1010
不变

pie=0100

pie=1001
图中是左移1位
二进制中是右移1位

na=1000

na= 1100
图中是右移1位
二进制中是左移1位

总结：当某一行做出选择后，进入到下一行，col不变，pie右移1位，na左移1位

52. N皇后 II-如何寻找当前行还剩哪些列可以选择

在第2行做选择时col,pie,na的限制条件为： col=1010 pie=0100 na=1000

由此可知，当前第2行仅剩第0列可供选择，如何计算得到呢？ ~（col | pie | na） & ((1<<n) -1)

1010
| 0100
| 1000
1110

~，取反 → **0001** → & ((1<<n) -1) → **bits=0001**

目前有1的二进制位代表无法选择

有1的二进制位代表可以选择

n为4时为1111

只保留n位多余的舍掉

注意：这里得到的结果中1可能不止一个
代表该行可选择的列不止一列
比如：bits=0010 0010

从bits中选择一位，然后进行下一行（选择最低位的1），如何做？ x & (−x) x & (x−1)

bits & (−bits) → **mask=0001** → 选择该列 → col | mask, pie | mask, na | mask → col=1011, pie=0101, na= 1001 → 更新二进制数进入下一行 → col不变, pie右移1位, na左移1位

获得bits的二进制表示中最低位的 1 的位置

bits= 0010 0010 时
mask= 0000 0010

bits & (bits-1) → bits=0000

将bits的二进制表示中最低位的1置成0

注意：当bits为0时代表这行的所有选择均已尝试完毕

52. N皇后 II-代码片段

```
class Solution {
    int total = 0;
    //终极解法：位运算
    public int totalNQueens(int n) {
        dfs(n,0,0,0,0);
        return total;
    }
    public void dfs(int n,int row,int col,int pie,int na) {
        if (row == n) {
            total++;
            return;
        }
        //算出棋盘当前行还有哪些位置可以放置皇后,由 0 变成 1，以便进行后续的位遍历
        int bits = ~(col | pie | na) & ((1<<n )-1);
        while (bits > 0) { //只要bits中还有1表明还有列可以放皇后
            int mask = bits & -bits; //取出最低位的1，在对应的列放皇后
            bits &= bits -1; //最低位的1对应的列放皇后了,消掉
            dfs (n,row+1,col | mask,(pie | mask )>>1,(na | mask) <<1) ;
        }
    }
}
```

其他题目

51. N 皇后

面试题 08.12. 八皇后

338. 比特位计数

面试题 16.07. 最大数值