棋盘中的棋盘

——浅谈棋盘的分割思想

上海市复旦附中　　俞鑫

引言

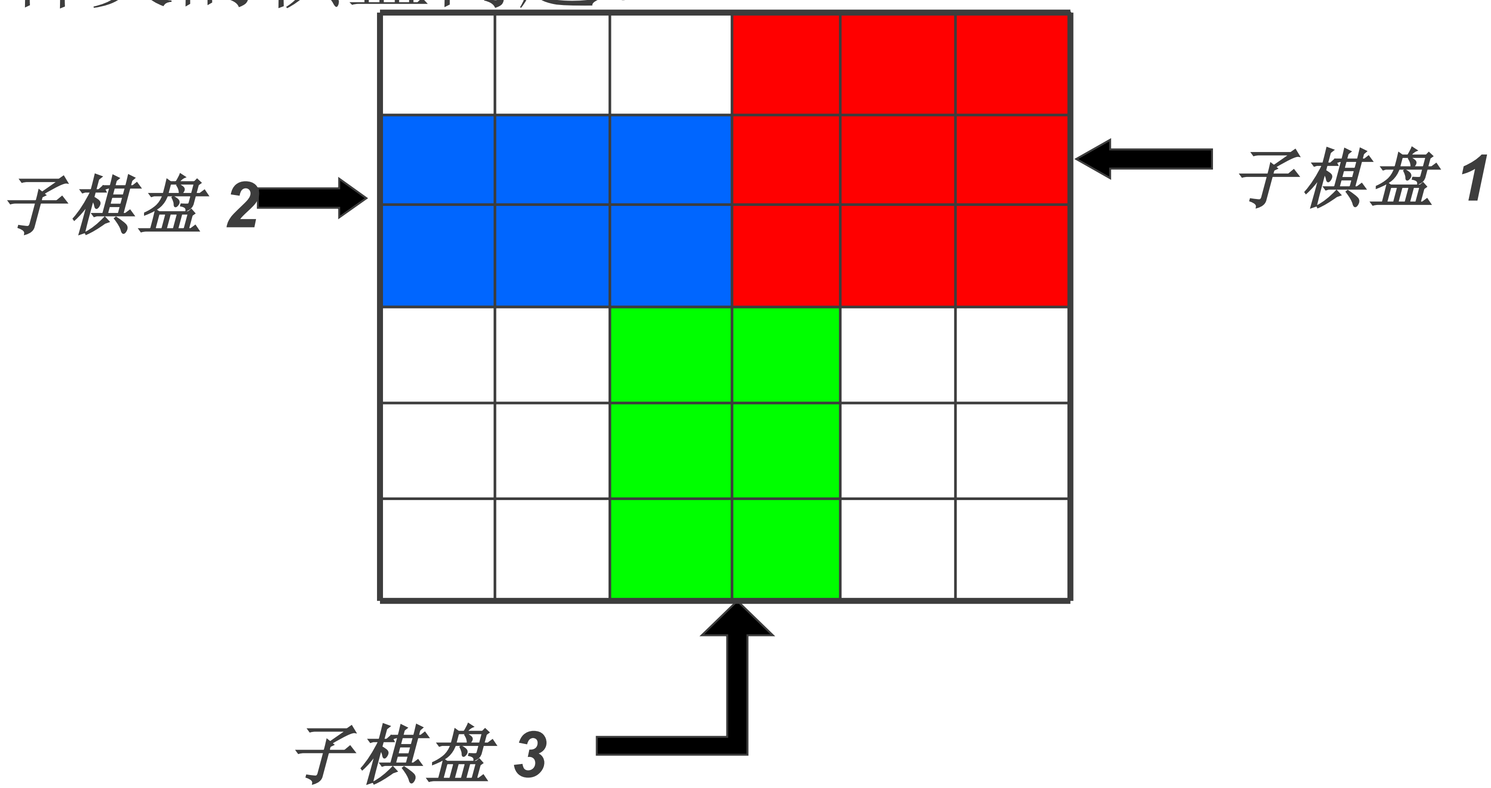
信息学是一门综合性的学科，也是一门充满乐趣的学科。

棋盘，作为一个重要的数学模型，以其趣味性和复杂的数学特性经常受到出题者的青睐。

因此，深入研究棋盘中蕴含的算法思想对于一名信息学爱好者而言是十分必要的。在此，我将着重说明棋盘中的一种重要思想——棋盘的分割思想。

引言

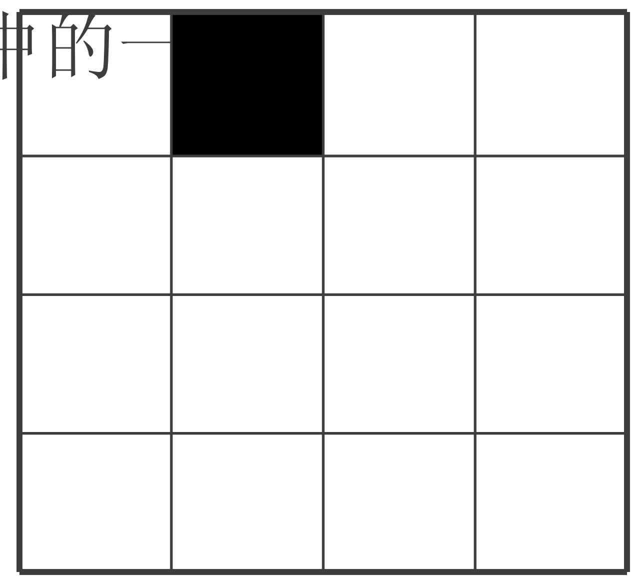
对于一个 m×n 的棋盘，它所含的子棋盘共有 $C_m^2 \times C_n^2$ 个，而其分割方法更是不计其数。巧妙地对棋盘进行分割，可以解决许多种类的棋盘问题。



例一：棋盘覆盖（经典问题）

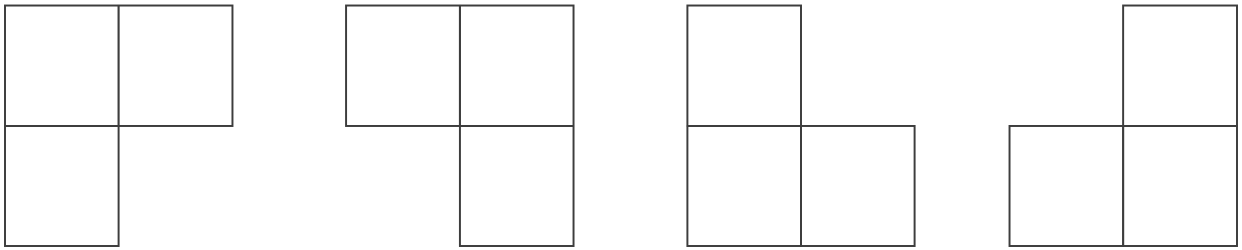
题目描述：

在一个 $2^k \times 2^k$ 方格组成的棋盘中，若恰有一个方格与其他方格不同，则称该方格为一特殊方格，且称该棋盘为一特殊棋盘。显然特殊方格在棋盘上出现的位置有 4^k 种情形。因而对任何 $k \geq 0$，有 4^k 种不同的特殊棋盘。图中的特殊棋盘是当 $k=2$ 时 16 个特殊棋盘中的一



在棋盘覆盖问题中，我们要用以下 4 种不同形态的 L 型骨牌覆盖一个给定的特殊棋盘上除特殊方格以外的所有方格，且任何 2 个 L 型骨牌不得重叠覆盖。

现求一种覆盖方法。



4 种不同形态的 L 型骨牌

输入：第一行为 k （棋盘的尺寸），第二行为 x,y ($1 \le x,y \le 2^k$)，分别表示特殊方格所在行与列。

输出：共 2^k 行，每行 2^k 个数，分别表示覆盖该格的 L 型的编号（特殊格用 0 表示）。

样例：

输入：

2

1 2

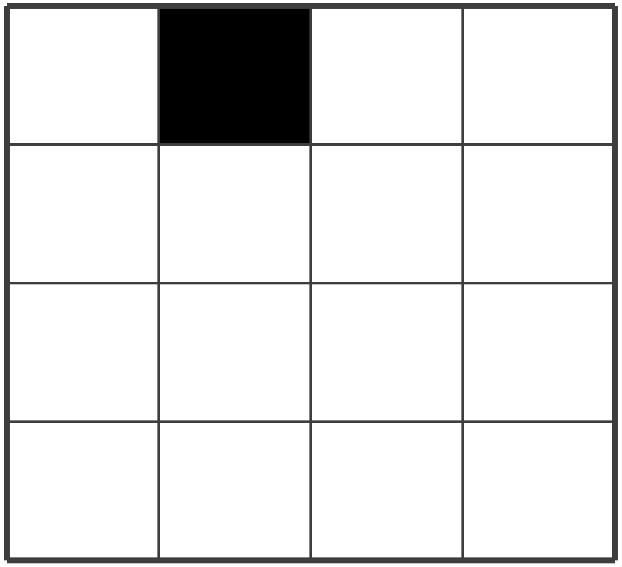
输出：

1 0 2 2

1 1 3 2

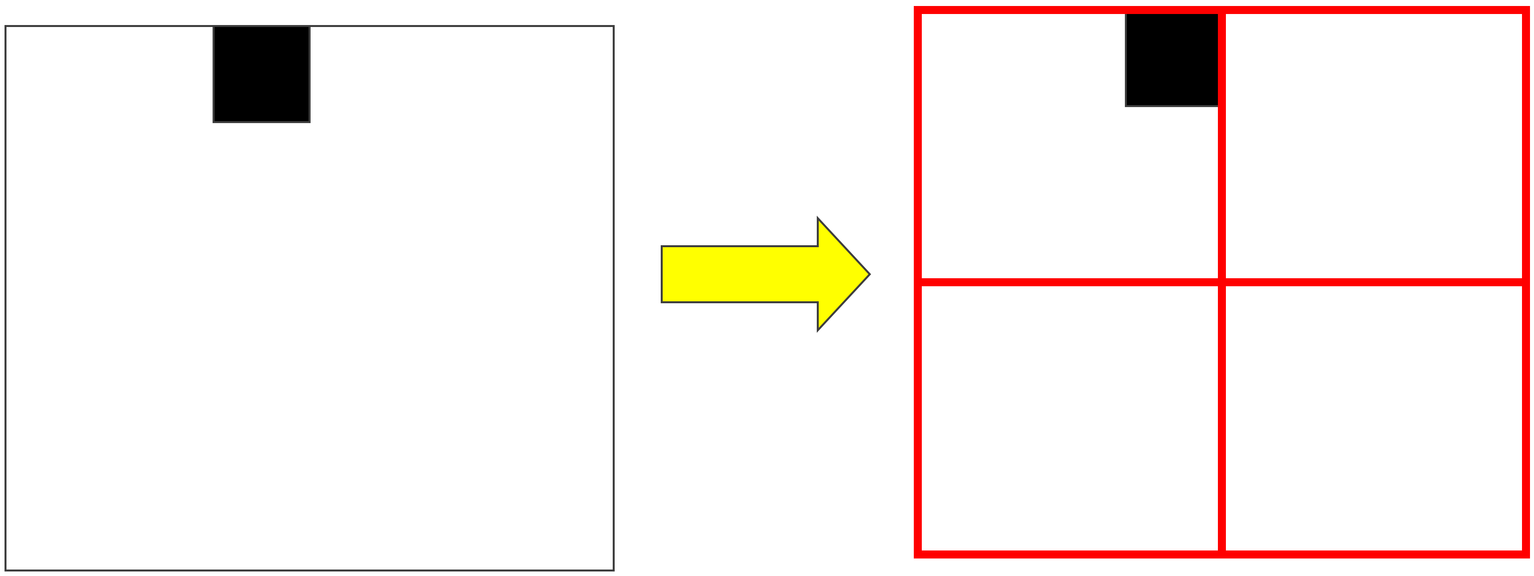
4 3 3 5

4 4 5 5



算法分析

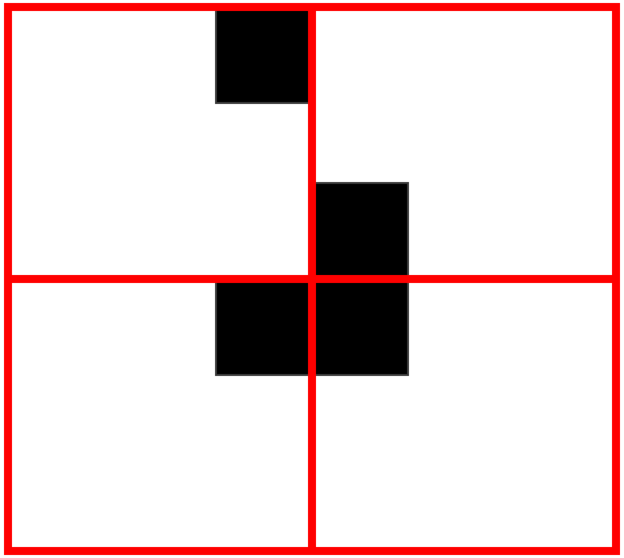
由棋盘尺寸为 $2^k \times 2^k$，我们可以想到将其分割成四个尺寸为 $2^{k-1} \times 2^{k-1}$ 的子棋盘



可是，由于含特殊方格的子棋盘与其它子棋盘不同，问题还是没有解决。

算法分析

只要稍作思考，我们就可以发现，只要将 L 型如图放置在棋盘的中央，就可以使四个子棋盘都变成特殊棋盘。此时问题也变成了四个相同的子问题，只需运用简单的递归就可以解决这道问题了。



程序实现

二位数组 num：覆盖该格的 L 型的编号，下文所说的对方格赋值即对其对应的 num 赋值。

x1,y1：当前棋盘左上角方格的行号与列号

x2,y2：当前棋盘右下角方格的行号与列号

x3,y3：当前棋盘中特殊格的行号与列号

ck：当前棋盘的尺寸（ $2^{ck}\times2^{ck}$ ）

cnum:当前 L 型骨牌的编号

初始值:

x1	y1	x2	y2	x3	y3	ck	cnum
1	1	2^k	2^k	x	y	k	1

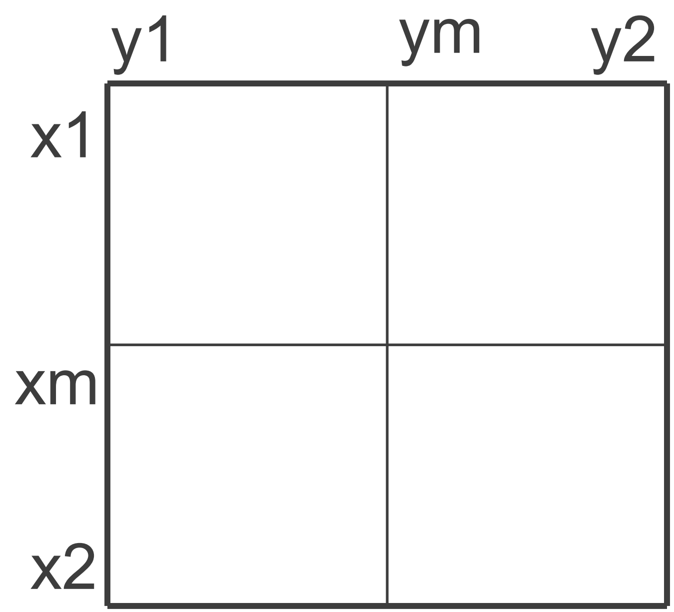
程序实现

开始时，将 num[x,y] 设为 0

当 ck=0 时：棋盘尺寸为 1×1 ，该格为已赋值的特殊格，不进行任何操作。

当 ck>0 时：设 xm 为 (x1+x2+1)/2 ， ym 为 (y1+y2+1)/2

比较 x3 与 xm ， y3 与 ym 的大小就能知道特殊格所在子棋盘的位置，将另外三个子棋盘中靠近棋盘中央的三个方格赋值为 cnum ，并分别作为这三个子棋盘的特殊格。随后 cnum 增加 1 。再对这四个棋盘分别进行处理。



复杂度分析

时间复杂度： $O(4^k)$

空间复杂度： $O(4^k)$

由于覆盖一个 $2^k \times 2^k$ 棋盘所需的 L 型骨牌个数为 $(4^k-1)/3$，故该算法是一个在渐进意义下最优的算法。

小结

将棋盘分割成子棋盘，要遵循以下两点：

1. 分割出的棋盘要与原棋盘尽可能相像。
2. 将原棋盘分割后尽量不要留下剩余部分。

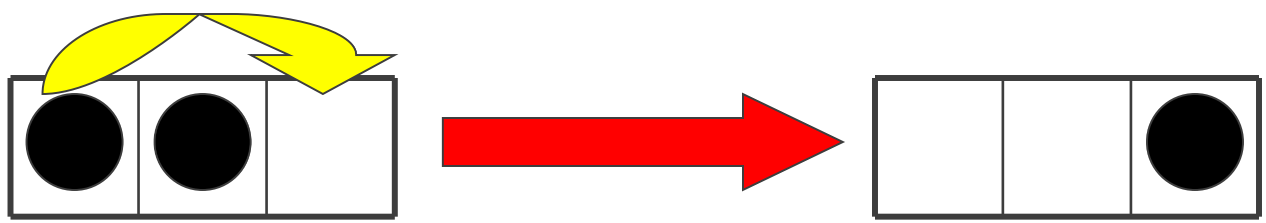
但如果分割后必定留下剩余部分又该如何呢

下面这道例题就是用来解答这个问题的。

例二：孔明棋问题（URAL1051）

题目描述：

在一个无限大的棋盘的格子上有一些棋子，这些棋子构成一个 m×n 的矩形 (m 为高度， n 为宽度且 1≤m,n≤1000）。你可以用一个棋子跳过另一个相邻的棋子，被跳过的棋子将被移去，请你求出最少能剩下几个棋子。



一种移动方法

输入： m,n

输出：最少能剩下的棋子数

样例：

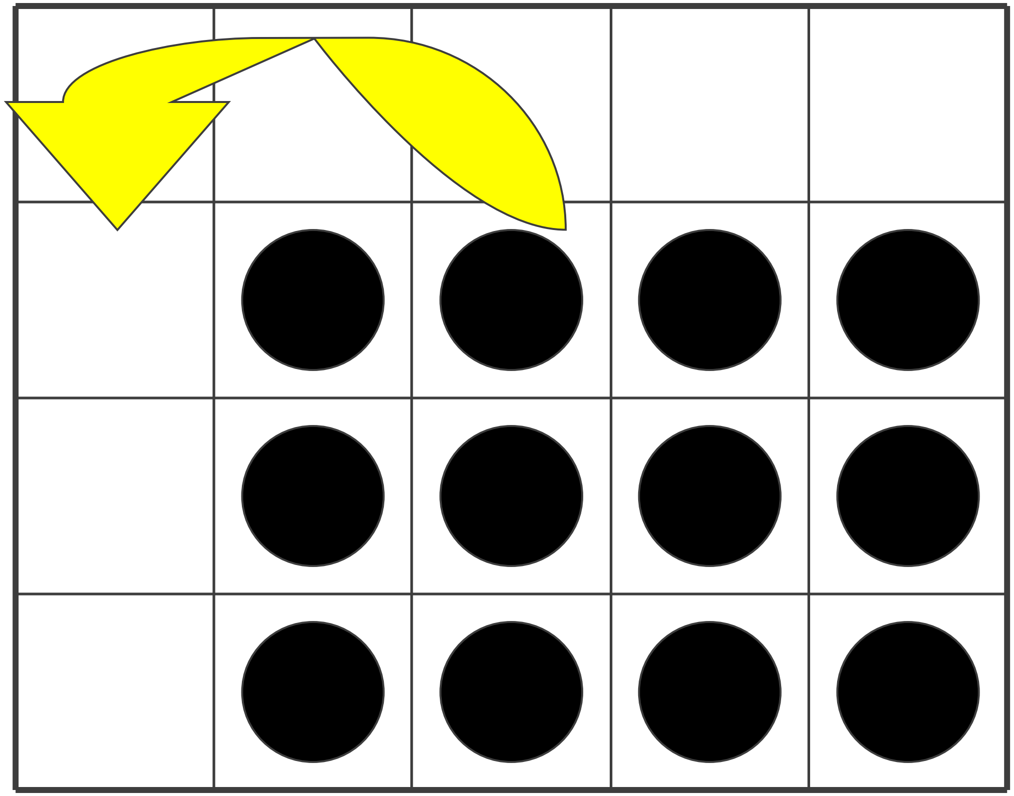
输入：

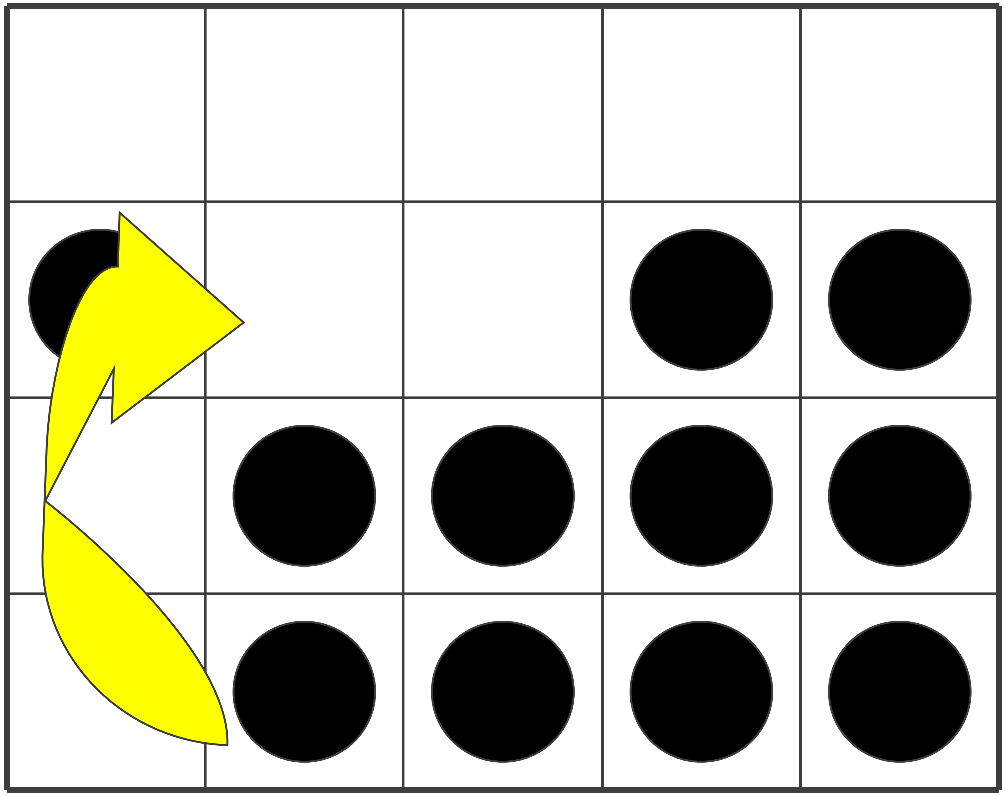
3 4

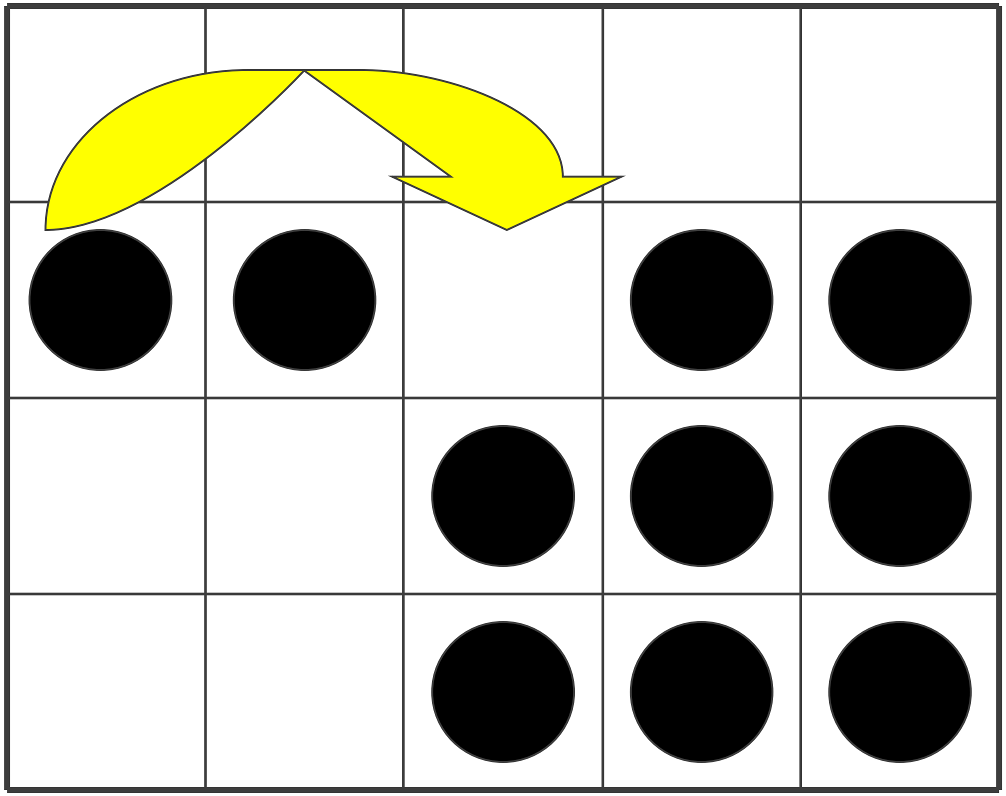
输出：

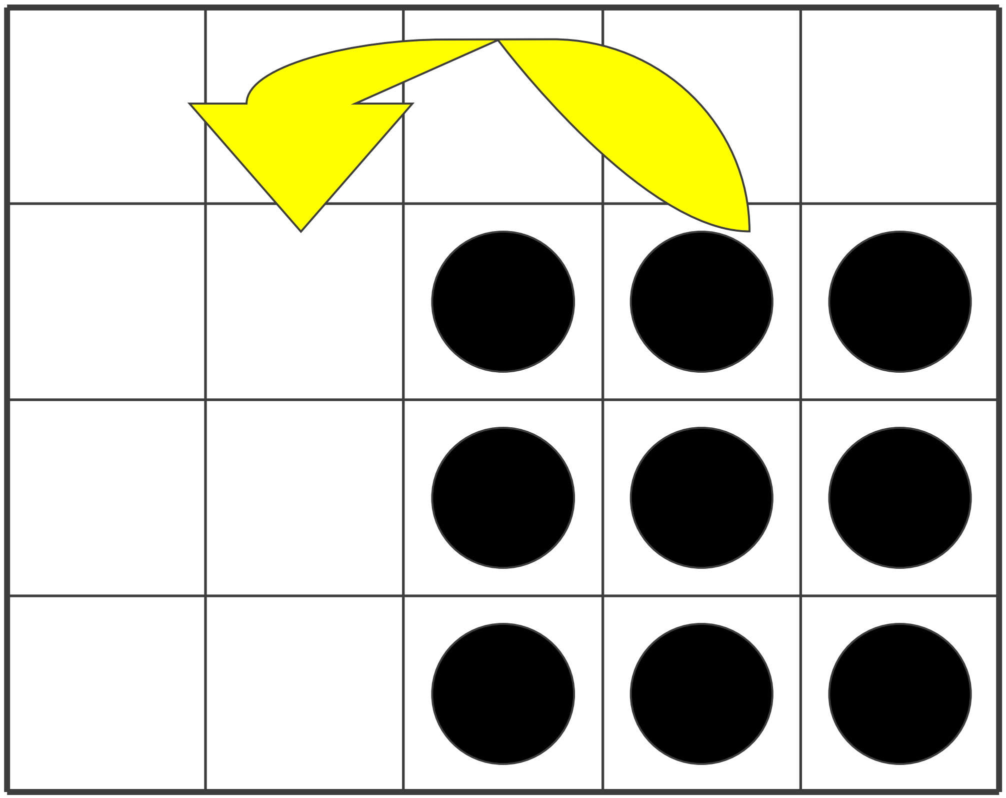
2

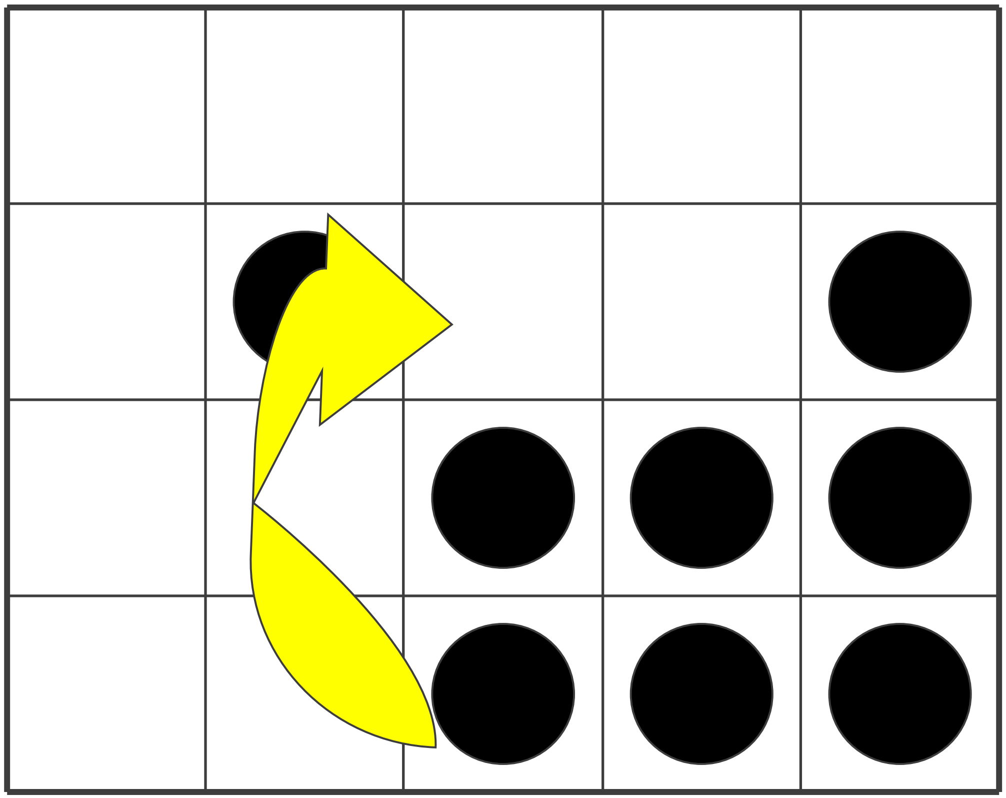
下面是样例的一种走法：

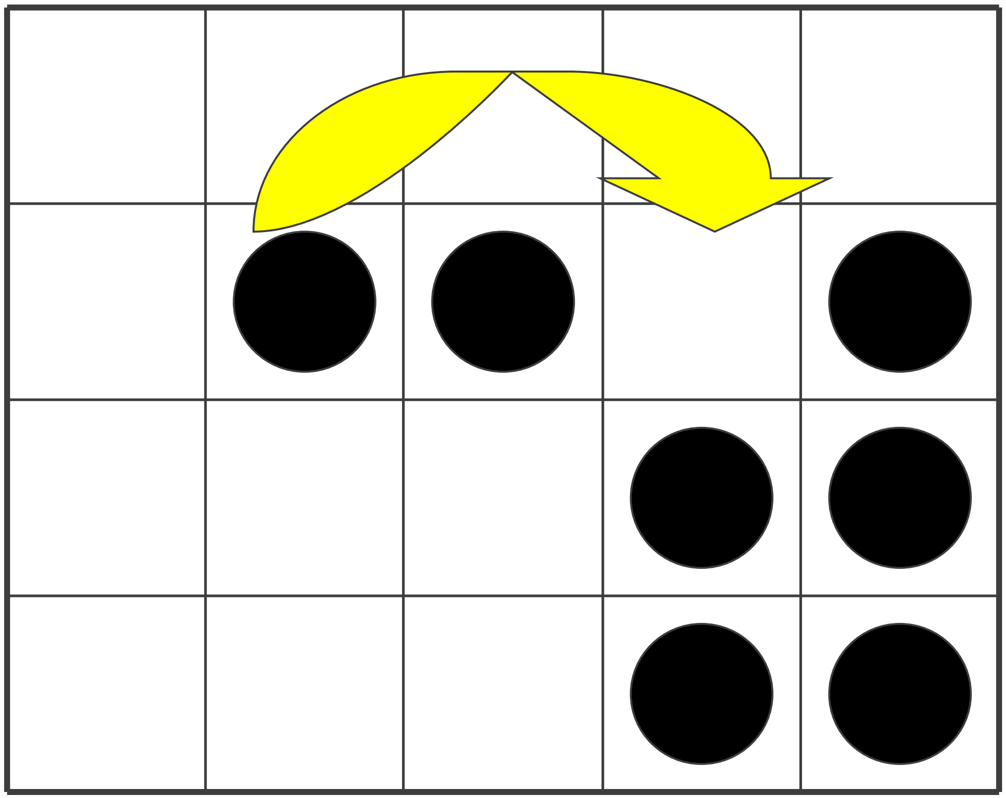


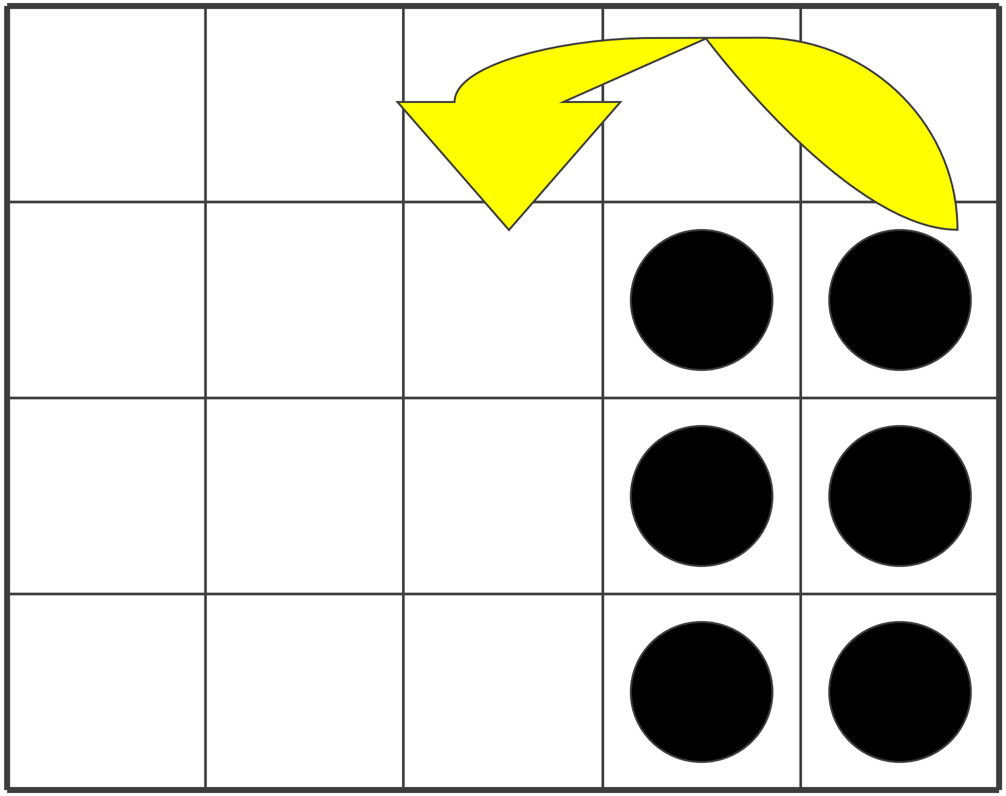


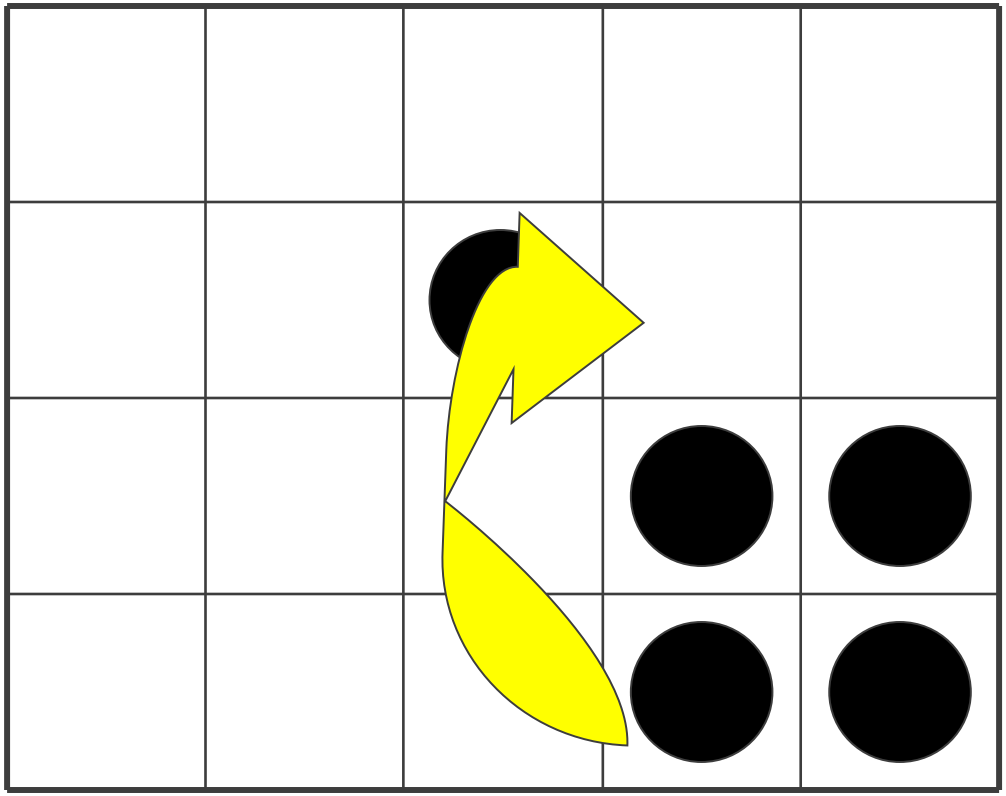


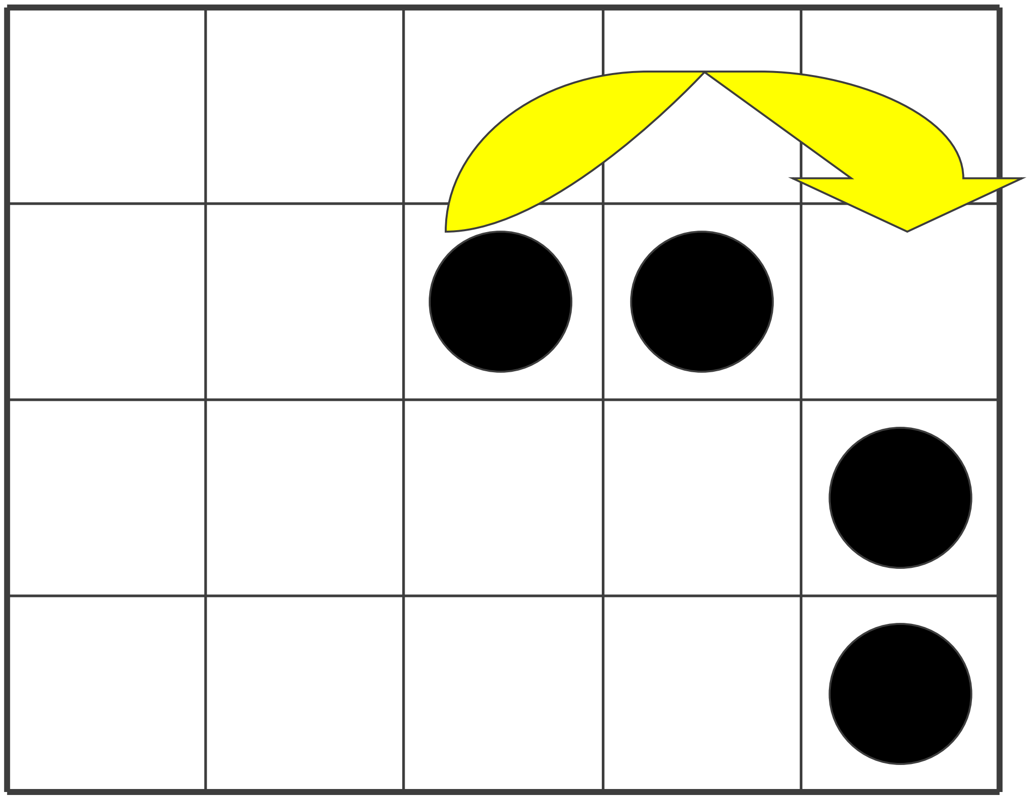


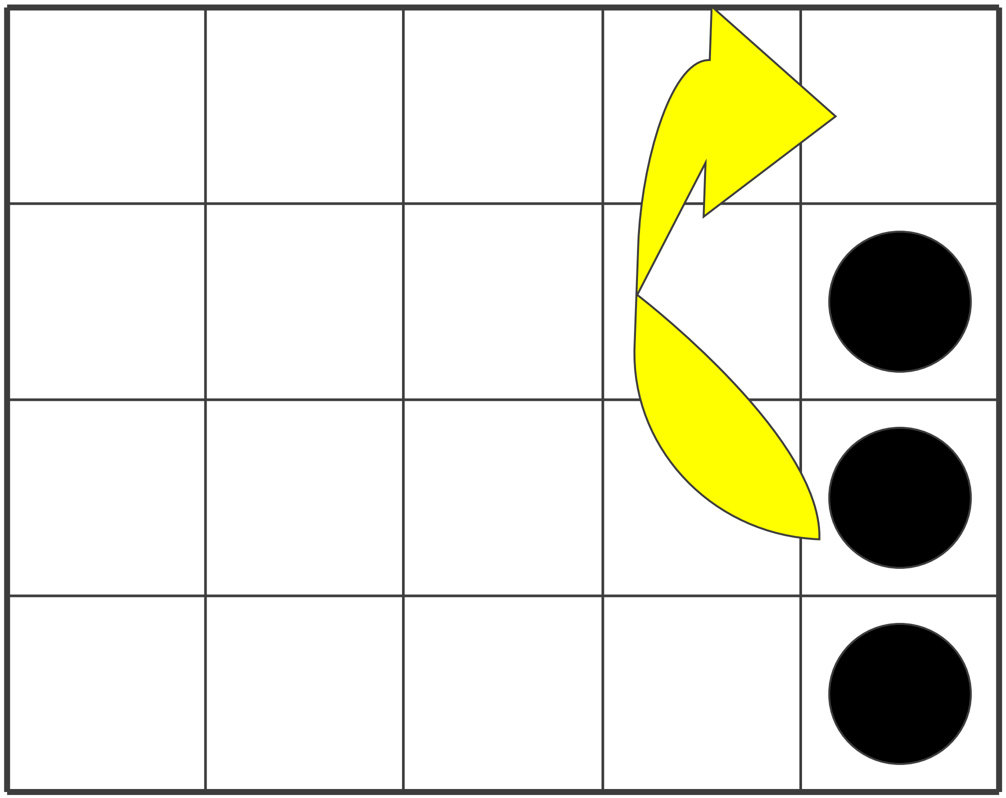


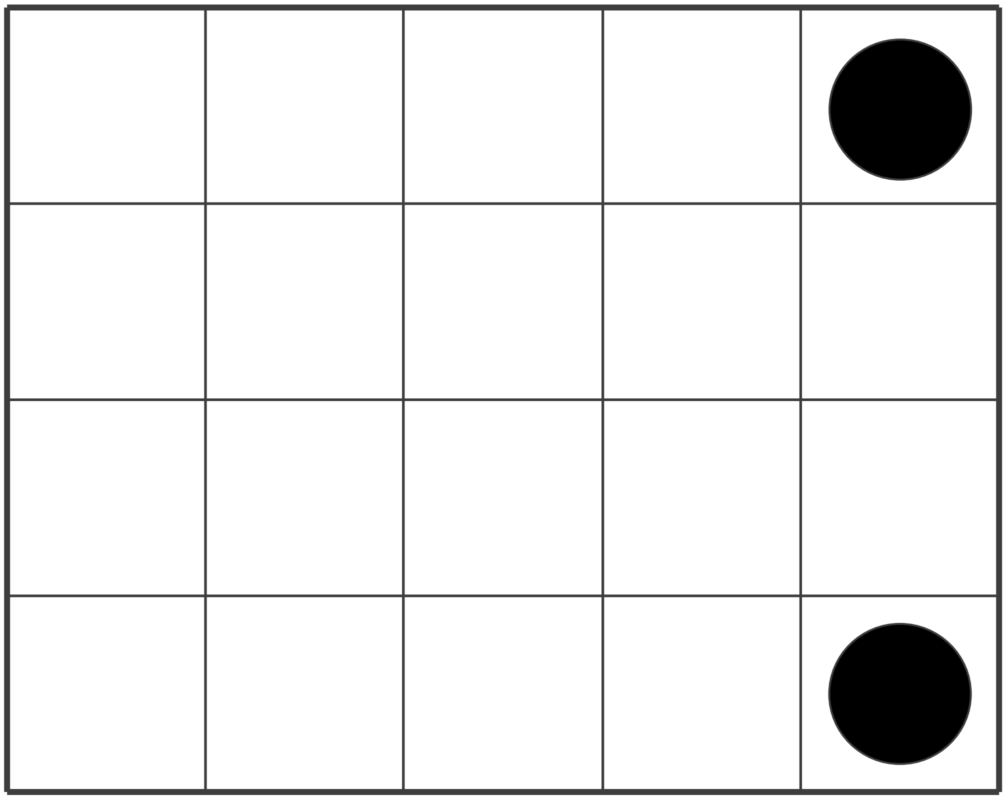






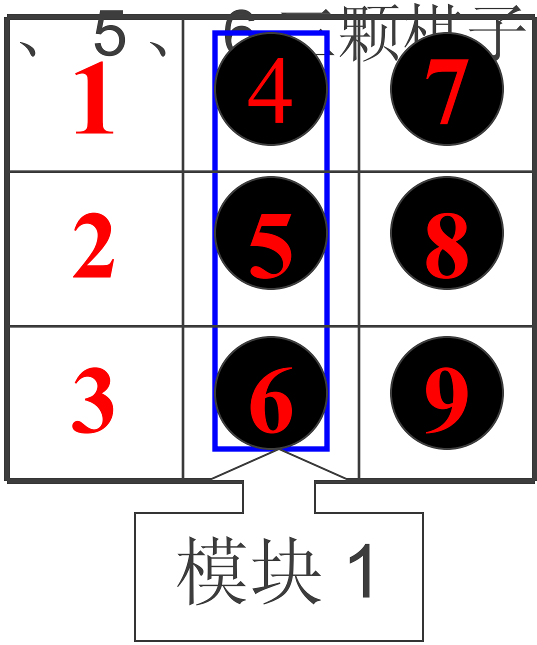






算法分析

由于 m=1 或 n=1 的情况比较特殊，我们先处理 m,n≥2 的情况。为了叙述方便，我们称由棋子所在格子组成的棋盘为“真棋盘”。通过样例，我们可以发现，对于图 (a) 中位于 4 、 5 、 6 格的连续三个棋子，若第 1 、 2 、 3 格上无棋子而第 7 、 8 、 9 格上均有棋子的话，则可以通过图 (b) 的操作将这三个棋子移去。我们称 4 、 5 、 6 三颗棋子为模块 1 。



图

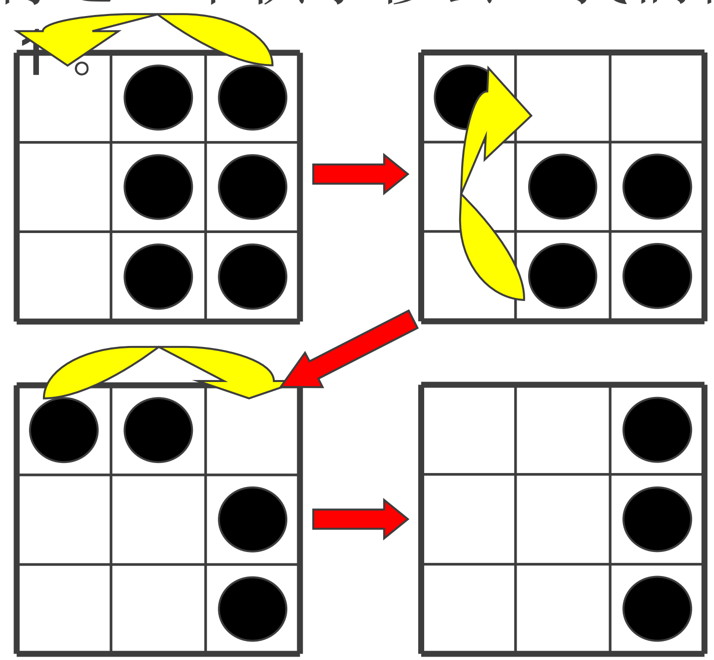
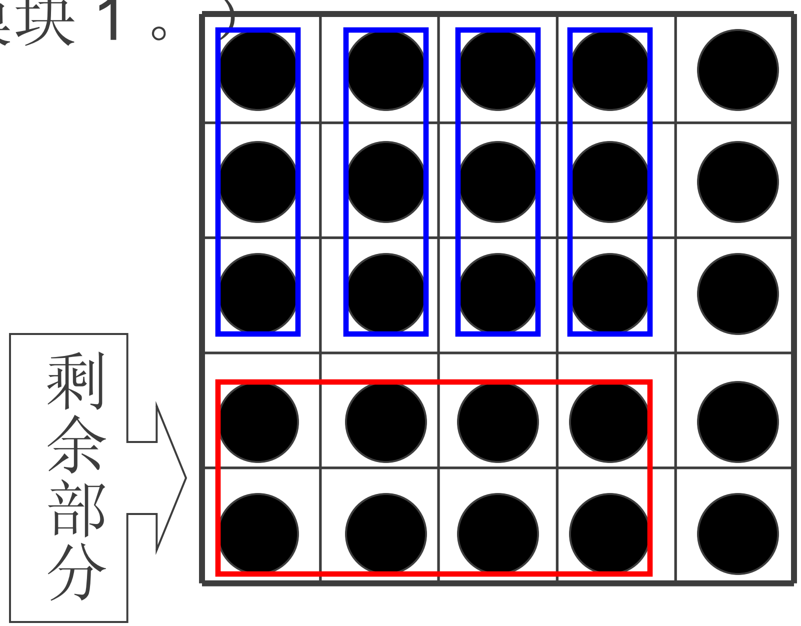


图 (b)

算法分析

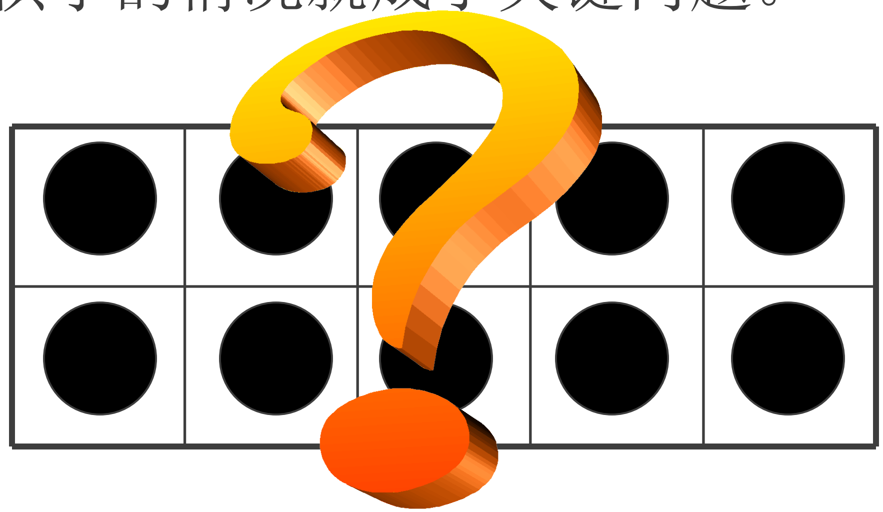
但是经过一些尝试后，我们发现只使用模块 1 对 m×n 的真棋盘进行分割效果并不理想。原因在于模块 1 每次对连续 3 行同时进行处理，当 m 不是 3 的倍数时，分割后总会留下剩余部分。（必须注意的是，图中用蓝框框起来的部分，必须等到其左边的棋子被去除后，才能成为模块 1 。）



因此，我们需要对剩余部分进行处理。

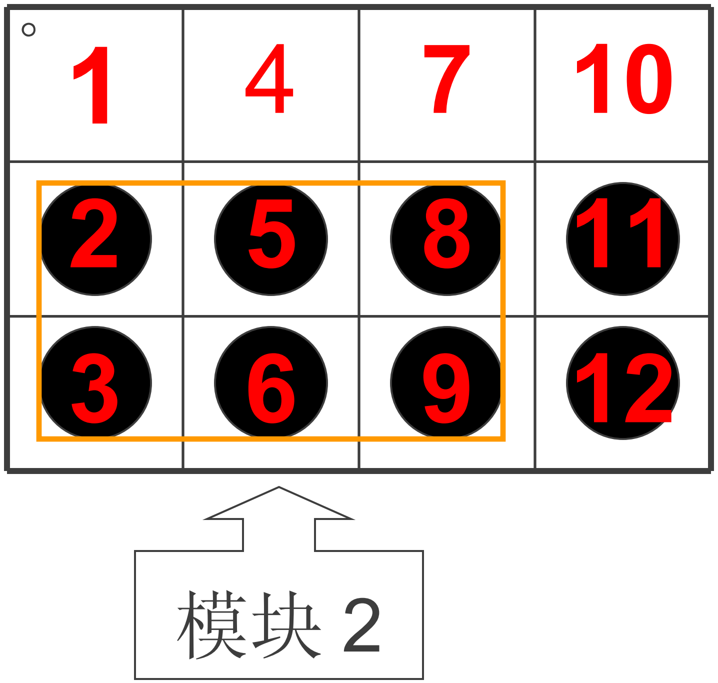
算法分析

我们发现，当 m 不为 3 的倍数时，总是留下 1 行或 2 行剩余部分。由于 1 行棋子很难去除，当只留下 1 行时，因为 $m \geq 2$，且 m 模 3 余 1，故 m 至少为 4 。于是我们就将最上方 4 行都作为剩余部分，对于剩下的 m-4 行，由于 m-4 是 3 的倍数，这 m-4 行可以用模块 1 进行分割。而 4 行棋子又可分为两部分，每部分都是 2 行棋子。因此，处理 2 行棋子的情况就成了关键问题。



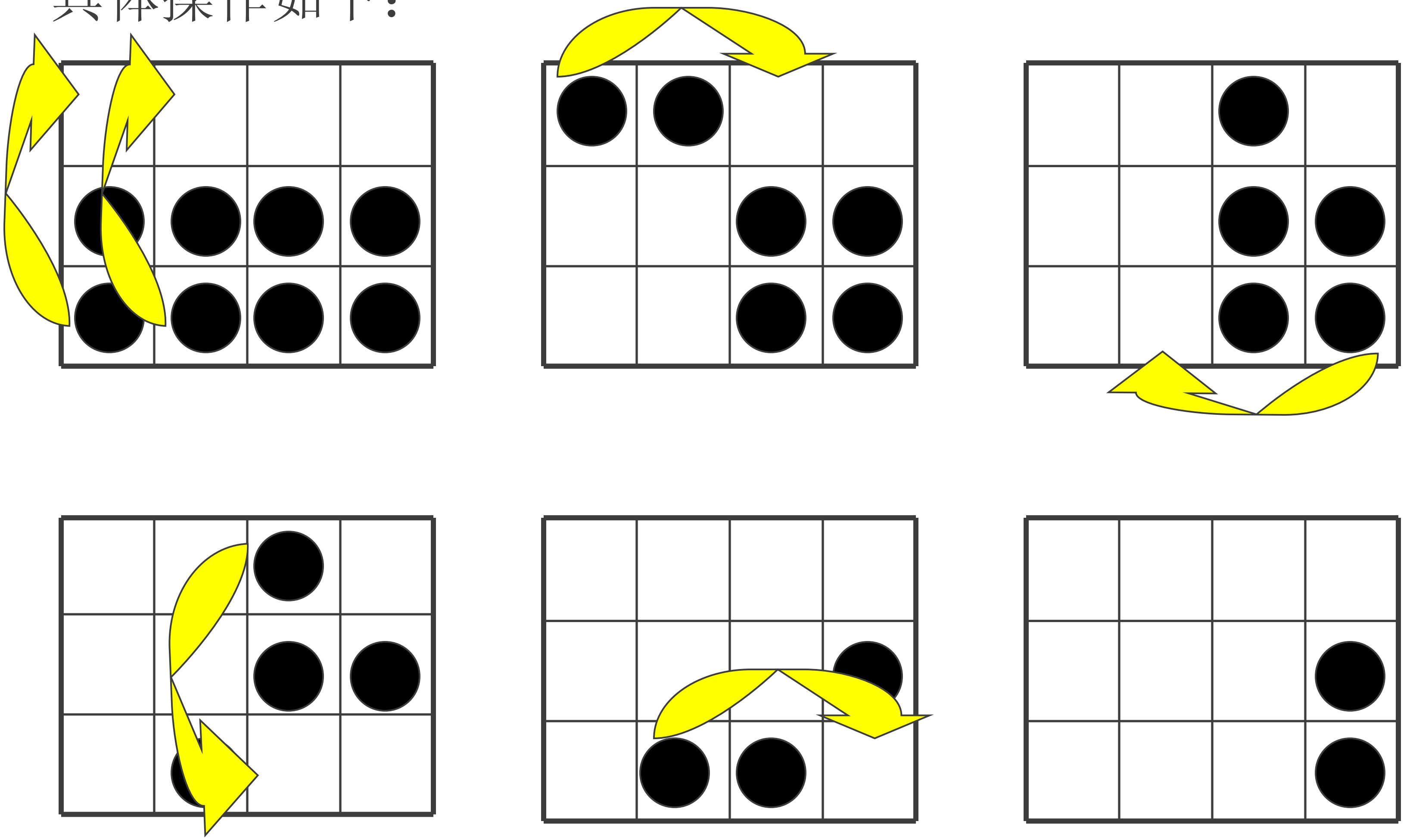
算法分析

经过尝试，我们发现对于图中第2、3、5、6、8、9格上的六个棋子，若第1、4、7格无棋子且第11、12格上有棋子的话，则可通过一系列的操作将这六颗棋子去除。我们称这六颗棋子为模块2。



算法分析

具体操作如下：



算法分析

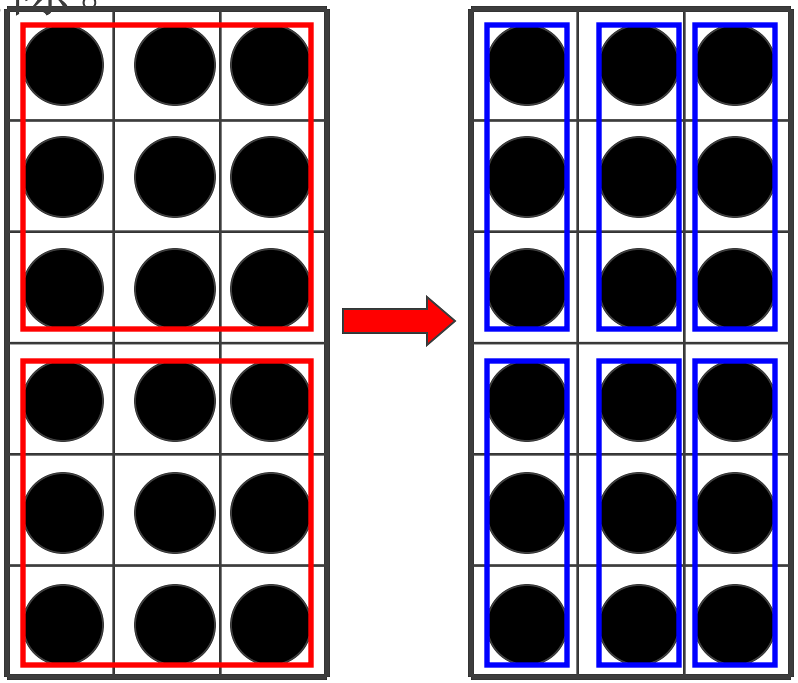
有了模块 1 和模块 2 这两样工具，对 m×n 真棋盘的分割就得心应手了。对任意的 m×n 的真棋盘，当 n≥5 时，对于棋盘中最左边的 3 列棋子形成的 m×3 的棋盘，我们通过下面的操作将其去除。

1 、当 m 是 3 的倍数时

首先将左边 3 列分成 m/3 个 3×3 的子棋盘。

再将每个 3×3 的子棋盘分成 3 个 3×1 的子棋盘。

每次对最上方最左边的 3×1 子棋盘进行操作，由于其左方无棋子，可以保证它是模块 1，将其去除。



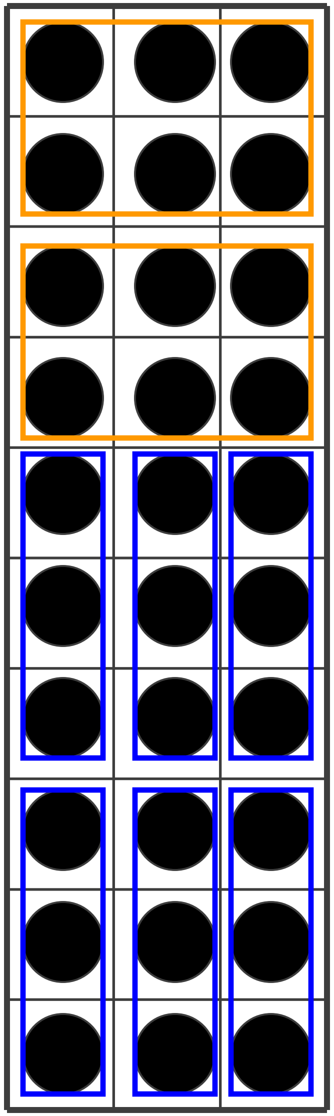
算法分析

2、当 m 模 3 余 1 时

由于 m≥2，故 m 至少是 4，$m\times3$ 棋盘最上方的 2×3 子棋盘是一个模块 2，将其去除。

$(m-2)\times3$ 棋盘最上方的 2×3 子棋盘也是一个模块 2，将其去除。

对于剩下的 $(m-4)\times3$ 棋盘，由于 m-4 是 3 的倍数，同 1 进行操作，将其去除。

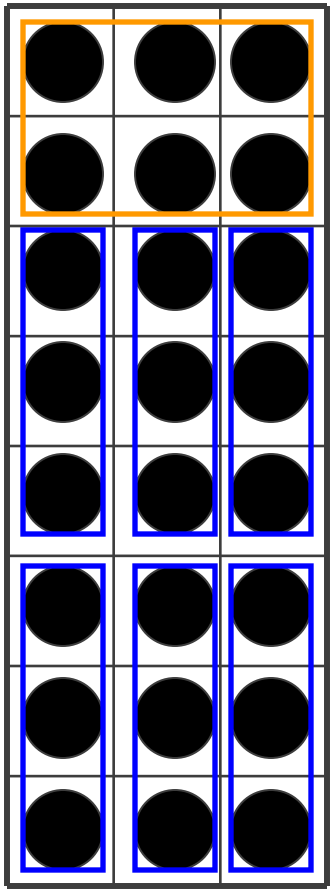


算法分析

3、当 m 模 3 余 2 时

m×3 棋盘最上方的 2×3 子棋盘是一个模块 2，将其去除。

对于剩下的 (m-2)×3 棋盘，由于 m-2 是 3 的倍数，同 1 进行操作，将其去除。



算法分析

设 m 与 p 关于 3 同余， n 与 q 关于 3 同余 ($2 \le p,q \le 4$)，对于任意的 $m \times n$ 的真棋盘，当 $n \ge 5$ 时，不断通过上述操作除去最左边 3 行，真棋盘规模将发生如下变化：

$$m \times n \rightarrow m \times (n-3) \rightarrow m \times (n-6) \rightarrow \ldots \rightarrow m \times q$$

再将棋盘顺时针旋转 90 度，真棋盘规模变为 $q \times m$，继续进行上述操作：

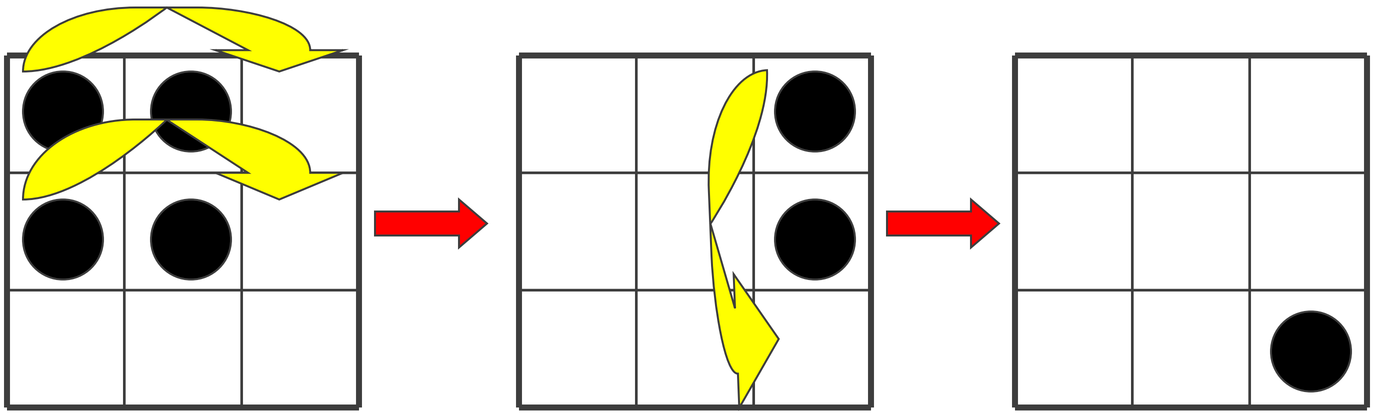
$$q \times m \rightarrow q \times (m-3) \rightarrow q \times (m-6) \rightarrow \ldots \rightarrow q \times p$$

再将棋盘顺时针旋转 90 度，真棋盘规模变为 $p \times q$。现在，我们只要对 $p \times q$ 的真棋盘进行操作，便可得到 $m \times n$ 的真棋盘经过操作最少能得到的棋子数的上界。

算法分析

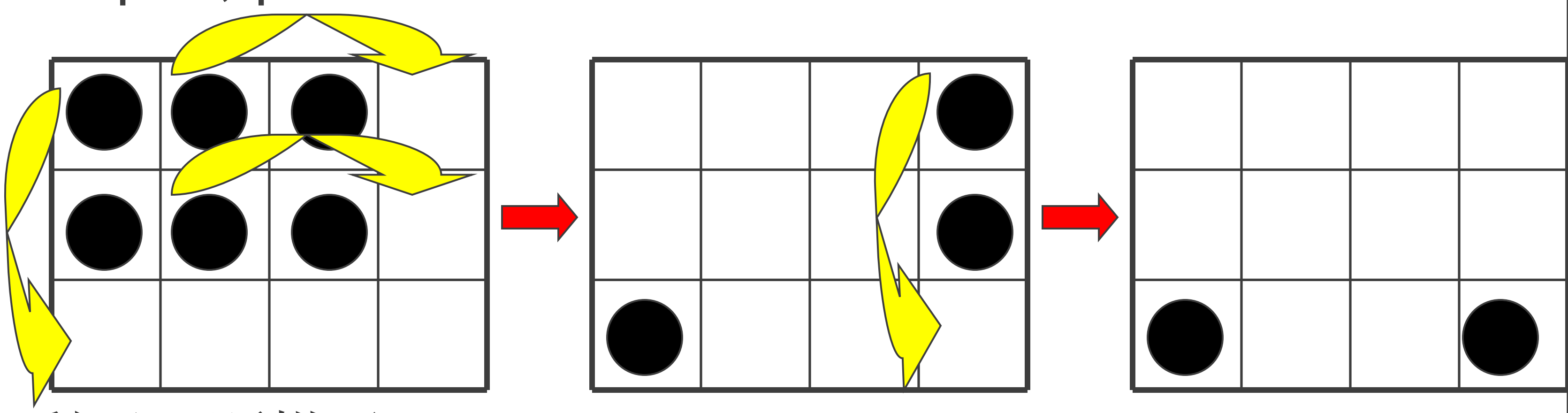
下面是对 p 、 q 各种可能情况的讨论：

① p=2,q=2



剩下 1 颗棋子

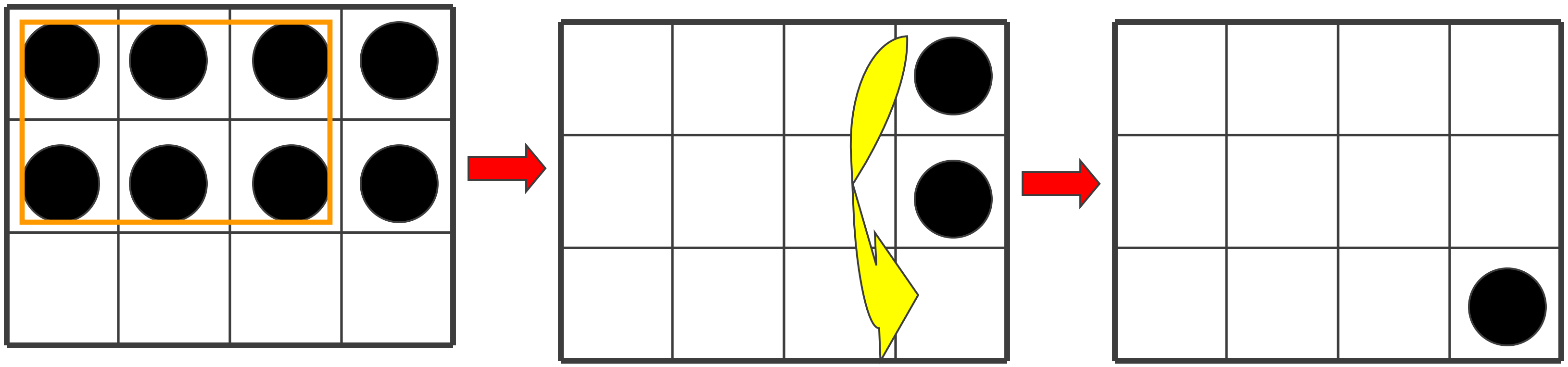
② p=2,q=3



剩下 2 颗棋子

算法分析

③ p=2,q=4



黄框中的 6 颗棋子构成模块 2，将其一起去除
◦剩下 1 颗棋子

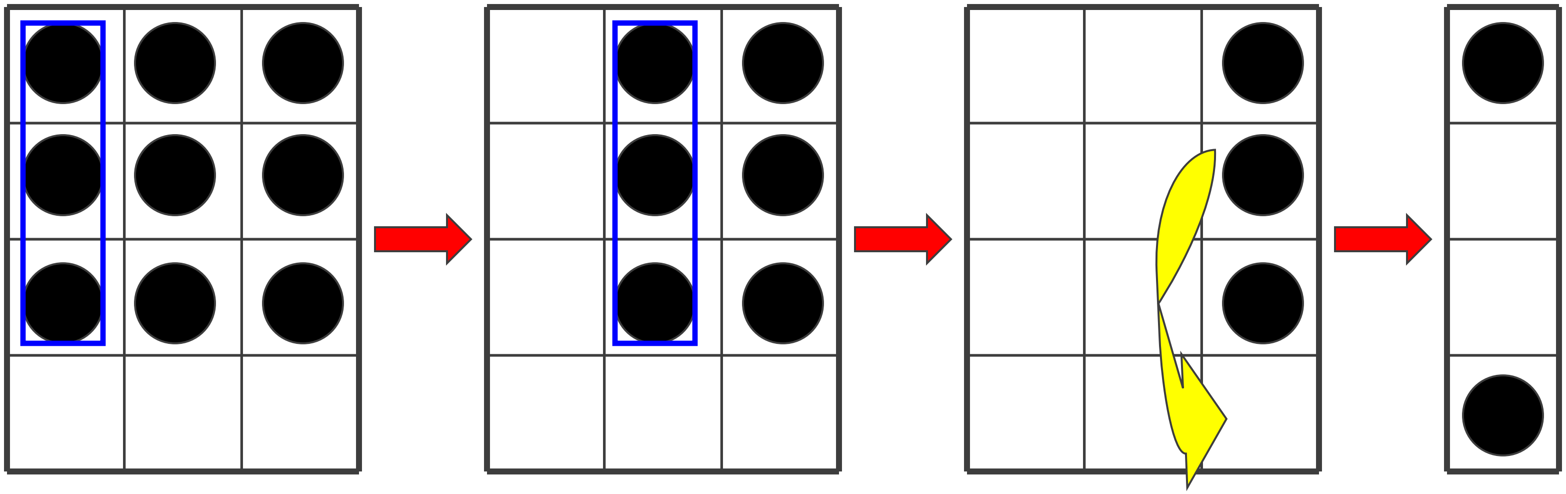
算法分析

④ p=3,q=2

将棋盘顺时针旋转 90 度后，与 p=2,q=3 的情况相同。

剩下 2 颗棋子

⑤ p=3,q=3



蓝框中的 3 颗棋子构成模块 1，将其一起去除。

剩下 2 颗棋子

算法分析

⑥ p=3,q=4

由样例的操作知，剩下 2 颗棋子。

⑦ p=4,q=2

将棋盘顺时针旋转 90 度后，与 p=2,q=4 的情况相同。

剩下 1 颗棋子

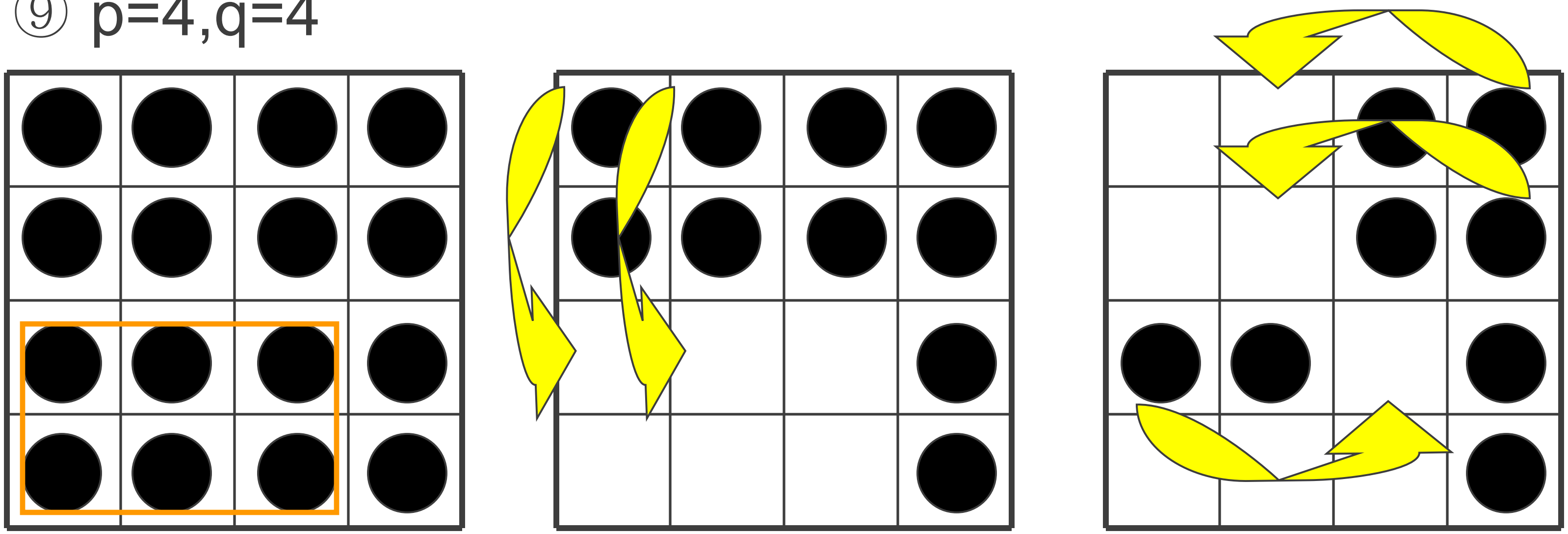
⑧ p=4,q=3

将棋盘顺时针旋转 90 度后，与 p=3,q=4 的情况相同。

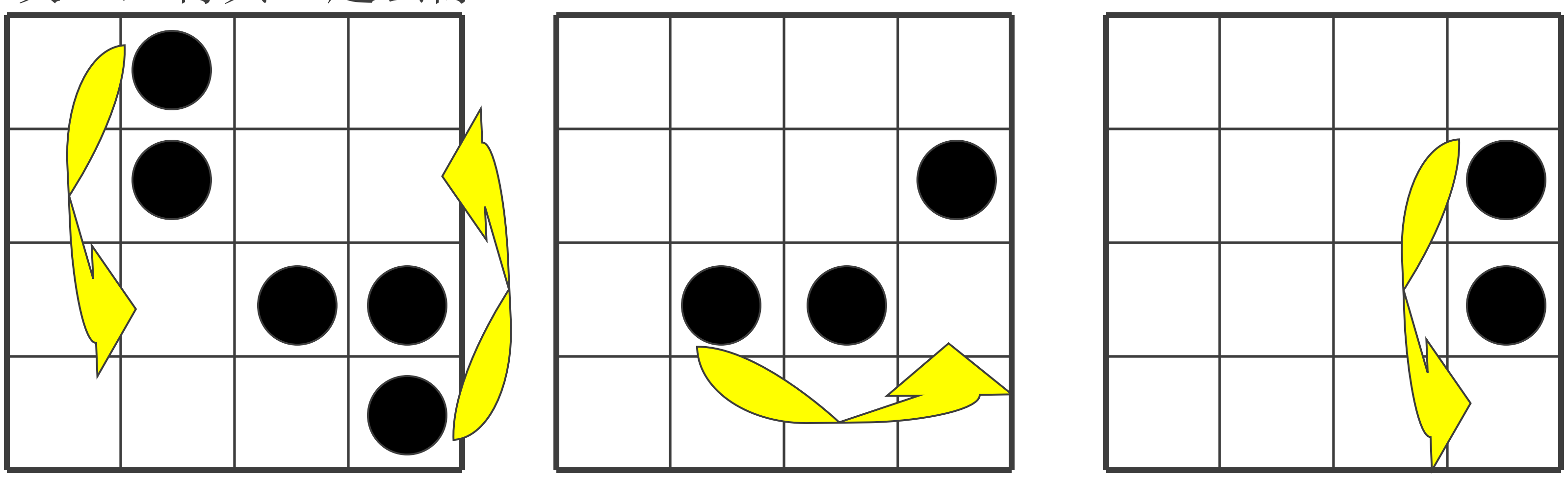
剩下 2 颗棋子

算法分析

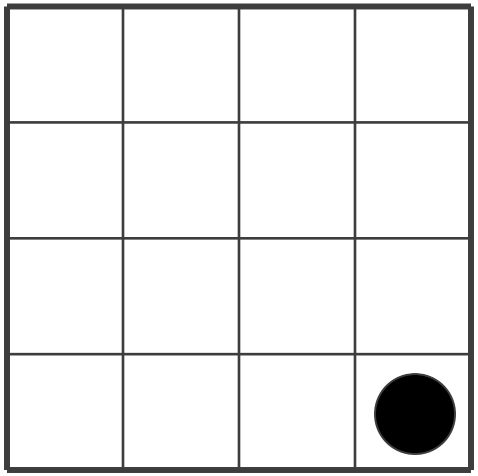
⑨ p=4,q=4



黄框中的 6 颗棋子构成模块 2，将其一起去除。



算法分析



剩下 1 颗棋子

算法分析

由于当 m 模 3 余 0、1、2 时，p 分别为 3、4、2。而当 n 模 3 余 0、1、2 时，q 分别为 3、4、2。我们可以列出下面这张表格来表示对各种情况的 m、n，m×n 的真棋盘经过操作最少能得到的棋子数的上界。

m mod 3 / n mod 3	0	1	2
0	2	2	2
1	2	1	1
2	2	1	1

为了叙述方便，我们设 m×n 的真棋盘经过操作最少剩下 s 颗棋子。

算法分析

经过观察可以发现，当 m 、 n 中有 1 个数是 3 的倍数时， s 的上界为 2 ，否则为 1 。

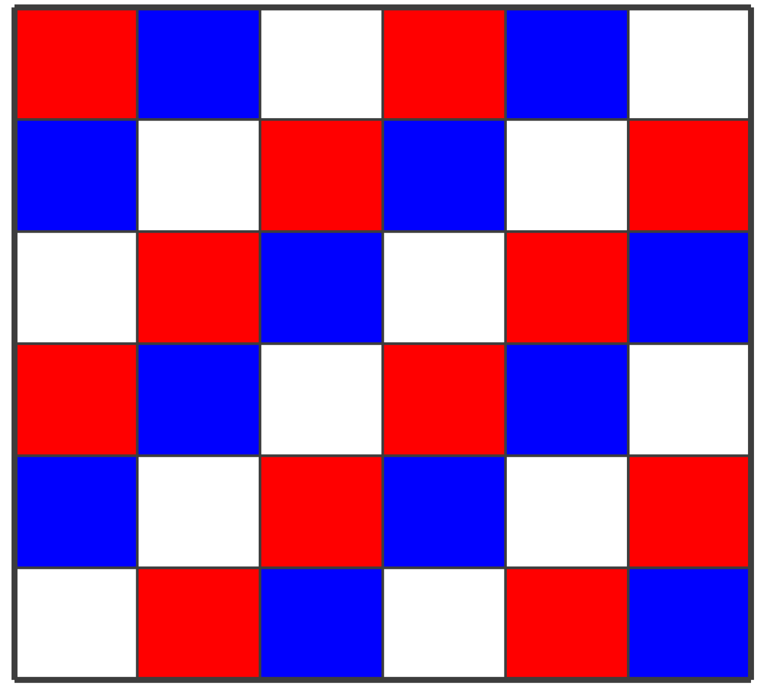
现在，我们只需要证明，这个上界就是答案。

(1) 对任意的正整数 m 、 n ，显然， $s \geq 0$ 。若 $s=0$ ，由于每次操作恰减少一颗棋子，所以最后一次操作前恰好剩余 1 颗棋子，但对于 1 颗棋子无论如何都不可能进行操作，矛盾。所以 $s \geq 1$ 。

(2) 当 m 、 n 中有一个数是 3 的倍数时，不妨设 m 是 3 的倍数（否则将棋盘顺时针旋转 90 度，即为 m 是 3 的倍数的情况），下面我们证明 $s \geq 2$ 。

算法分析

首先，我们对无限大的棋盘进行三色间隔染色。其中，无论横排还是竖排的任意 3 个连续方格，必为一红、一蓝、一白。对于 $m\times n$ 的真棋盘中的任何一列，都是一个 $m\times 1$ 的子棋盘，而每一个 $m\times 1$ 的子棋盘又可以分成 $m/3$ 个 3×1 的子棋盘。于是，



$m\times n$ 的真棋盘可以分为若干个 3×1 的子棋盘，而每个 3×1 的子棋盘中都有红、白、蓝方格各一个，所以，真棋盘中红、白、蓝方格的数目相同。设在红、白、蓝格上的棋子数分别为 a 、 b 、 c ，开始时 a=b=c 。

算法分析

对于每一次操作，都可以看作连续 3 个格中，两个格中的棋子被拿去，而另一格上多了一颗棋子，于是 a 、 b 、 c 中有两个数减 1 ，另一个数加 1 ，奇偶性都发生了变化，所以 a 、 b 、 c 始终同奇同偶。若 s=1 ，由于操作完成之后， s=a+b+c ，故 a 、 b 、 c 中两个为 0 ，一个为 1 ，奇偶性不同，矛盾。又因为 s≥1 ，所以 s≥2 。

通过上面的证明，我们得出了 s 的下界，它与先前的上界完全相同。于是，当 m,n≥2 时，若 m 、 n 中有一个数是 3 的倍数，则 s=2 ；否则， s=1 。

算法分析

当 m=1 或 n=1 时，通过几个实例，我们基本上可以猜测出最少剩下的棋子数。当 m=1 时， s 为 n-[n/2] ，当 n=1 时， s 为 m-[m/2] 。由于篇幅有限，详细论证请参考我的论文。

于是，最少剩下的棋子数 s 为：

$$s = \begin{cases} n-[n/2]\text{，} & \text{当 } m=1 \text{ 时} \\ m-[m/2]\text{，} & \text{当 } n=1 \text{ 时} \\ 2\text{，} & \text{当 } m,n\geq 2 \text{ 且 } m\text{、}n \text{ 中有一个数是 } 3 \text{ 的倍数时} \\ 1\text{，} & \text{当 } m,n \geq 2 \text{ 且 } m\text{、}n \text{ 中无 } 3 \text{ 的倍数时} \end{cases}$$

程序实现十分简单，在此不再赘述。

小结

在这道题中，将棋盘分割成模块 1 的想法遇到了障碍，其主要原因在于分割后总是留下剩余部分。为了处理留下的剩余部分，我们发现了模块 2 。其实，从本质上讲，模块 2 只不过是针对剩余部分的一些有规律的步骤。所以，当棋盘的分割出现困难，必须留下剩余部分时，可以采用以下两个办法：

1. 对剩余部分进行单独处理，可以用较少的特殊步骤（如最后对小棋盘的处理），或一些有规律的步骤（如模块 2 ）。

2. 如果剩余部分实在难以处理，我们必须放弃原来的分割方案，另起炉灶。

总结

作为处理棋盘的一种重要思想，棋盘的分割方法其实远不只如此。棋盘的分割思想从本质上说是一种递归的思想，但又不尽相同，它比纯粹的递归更加复杂，更加难以捉摸。只有平时多留心，多思考，才能掌握其中众多的奥妙，在实战中做到

博观而约取，厚积而薄发。

谢谢大家
！