链滴

《程序员的算法趣题》-Q03- 翻牌

作者：DattyRabbit

原文链接：https://ld246.com/article/1599490045580

来源网站：链滴

前言

此篇为《程序员的算法趣题》中的入门篇第3题"翻牌"的相关解题分析博文。

关于该系列的介绍请看：

《程序员的算法趣题》-开坑记录

ps:这个题目是现在碰到的最为简单的一个题了，感觉从看完题目到做完，是唯一真正的满足了作者给的10分钟的要求。:flushed:

题目

这里有 100 张写着数字 1~100 的牌，并按顺序排列着。最开始所有牌都是背面朝上放置。某人从第 2 张牌开始，隔 1 张牌翻牌。然后第 2,4, 6, ..., 100 张牌就会变成正面朝上。

接下来，另一个人从第 3 张牌开始，隔 2 张牌翻牌（原本背面朝上的，翻转成正面朝上；原本正面朝的，翻转成背面朝上）。再接下来，又有一个人从第 4 张牌开始，隔 3 张牌翻牌（ 图1 ）。

像这样，从第 n 张牌开始，每隔 n - 1 张牌翻牌，直到没有可翻动的牌为止。

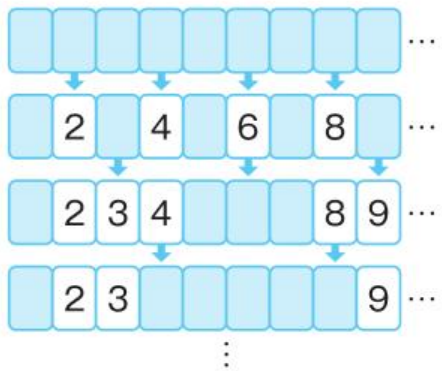
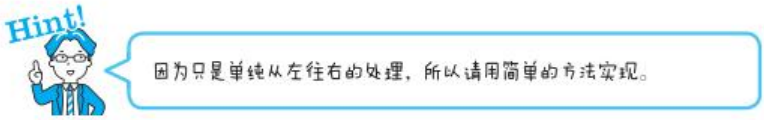


图1 翻牌示意图

问题

求当所有牌不再变动时，所有背面朝上的牌的数字。



作者思路及代码实现

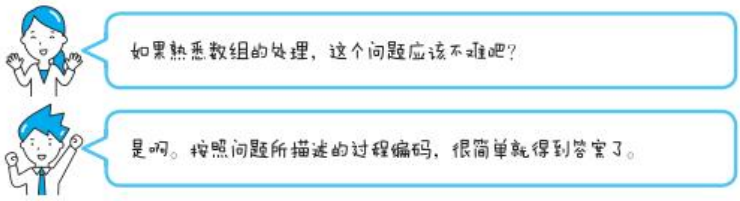
思路

只要根据问题描述，按顺序对牌进行翻转处理就可以了。用数组保存牌的状态，如果牌正面朝上，则置值为 true，反之为 false。这样一来，我们就可以简单地模拟翻转操作了。用 Ruby 时，可以用下这个程序来实现（代码清单 03.01）。

```
# 初始化卡牌
N = 100
cards = Array.new(N, false)

# 从 2 到 N 翻牌
(2..N).each{|i|
    j = i - 1
    while (j < cards.size) do
        cards[j] = !cards[j]
        j += i
    end
}

# 输出背面朝上的牌
N.times{|i|
    puts i + 1 if !cards[i]
}
```

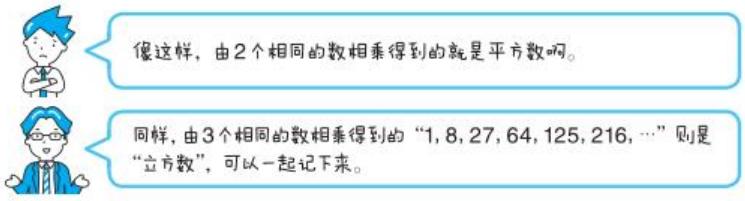


Point

代码清单 03.01 是用数组来实现的，但从左到右按顺序处理也就意味着“已经翻转过的部分不再翻转。如果针对这一点进行优化，还可以继续简化程序，具体如代码清单 03.02 所示。

```
(1..100).each{|i|
        flag = false
        (1..100).each{|j|
            if i % j == 0 then
                flag = !flag
            end
        }
        puts i if flag
    }
```

执行这个代码后就可以正确输出答案“1、4、9、16、25、36、49、64、81、100”。从答案可以到，结果都是“平方数”。



如果翻牌操作进行了奇数次，则最后是正面朝上；如果进行了偶数次，则最后是背面朝上。也就是说这个问题等价于“寻找被翻转次数为偶数的牌”。而翻牌操作的时机则是“翻牌间隔数字是这个数的数时”，因此也就相当于寻找拥有偶数个“1 以外的约数”的数字。

举个例子，12 的约数是“1、2、3、4、6、12”这 6 个，也就是偶数个。把约数由小到大排列，并

两端的数按顺序相乘就可以得到原数。

例） 1×12,　2×6,　3×4

不过 16 的约数是“1、2、4、8、16”这 5 个，也就是奇数个。我们把约数从小到大排列，并将两的数按顺序相乘后，会剩下正中间的数字 4。

例） 1×16,　2×8

※剩下的数字乘以自身就可以得到原数（4×4 = 16）

也就是说，只有当牌面数字是平方数的时候约数才是奇数个，也就是除 1 以外的约数是偶数个。了解这个规律后，即便不编程，也能知道答案。在日常工作中，动手编程之前最好也像这样好好想一想。

答案

1,　4,　9,　16,　25,　36,　49,　64,　81,　100

Column

讨厌麻烦的人比较适合做程序员吗

程序员是个非常有魅力的职业，他们写几行代码就能从零开始创造新的价值。从某种意义上说，这可称得上是“发明创造”。

大家有时候也会谈论，适合这种职业的究竟是什么样的人呢？提到程序员，大家通常会有理工科大学业、宅、喜欢游戏等印象。事实上，在编程开发的前线，文科出身的程序员还是挺多的，也有很喜欢动的程序员。

如果非要给出一个适合做程序员的条件，我的第一反应是“讨厌麻烦”这几个字，也就是不喜欢重复械的工作，希望尽可能地实现自动化。如果某个工作需要花费 30 分钟进行机械重复的操作，程序员能会为了瞬间完成工作而花费 1 个小时来编程实现。大概就是这种心境吧。

事实上，我学习编程的契机也是碰到了麻烦的事情。学生时期，老师告诉我，要想记住键盘上的键位就要不断地从 A 敲到 Z。要一直重复练习，直到让屏幕填满字母。

我很讨厌这种重复劳动，为了轻松一点，就编写了一个自动让屏幕填满 A 到 Z 的程序。保存好这个序之后，下一次再执行，就可以一瞬间将字母填满屏幕。

在那之后，每当遇到麻烦的事情，我就不断编写程序来解决，无形中练就各种编程技巧。我与编程因碎小事而邂逅，如今算来都已经 20 余年了。

自己做的思路及实现

看到这个题目的时候，稍微迟疑了一下什么叫做 "求当所有牌不再变动时，所有背面朝上的牌的数字。？然后为了确定一下我理解的对不对，就稍微往后看了下作者是怎么说的。虽然提前看了作者的解题但是我觉得也不冲突，因为这个题真的太简单了。我只是比较犹豫所谓的"当所有牌不再变动时"是不就是n大于100的情况。

然后看到确实是这样的，也就马上开始写代码实现了。我是用数组来模拟牌组的，数组下标就模拟牌号码，下标+1即为牌的数字，然后初始化一个长度为n的int数组，因为初始化int类型默认是0，所以按照值是0代表背面朝上的情况，然后值为1的时候代表正面朝上的情况。

然后翻牌的操作我就直接在循环到需要操作的数位的时候去进行一次与1的异或操作来模拟翻牌。最翻牌的操作遍历完成之后再遍历一遍输出值为0的位置的下标+1就可以完成。

话不多说上代码

```
/**
 * 《程序员的算法趣题》 - 入门篇 - Q03 - 翻牌
 *
 * 题目：这里有 100 张写着数字 1~100 的牌，并按顺序排列着。最开始所有
 * 牌都是背面朝上放置。某人从第 2 张牌开始，隔 1 张牌翻牌。然后第 2,
 * 4, 6, ..., 100 张牌就会变成正面朝上。
 * 接下来，另一个人从第 3 张牌开始，隔 2 张牌翻牌（原本背面朝上
 * 的，翻转成正面朝上；原本正面朝上的，翻转成背面朝上）。再接下来，
 * 又有一个人从第 4 张牌开始，隔 3 张牌翻牌（ 图1 ）。
 * 像这样，从第 n 张牌开始，每隔 n - 1 张牌翻牌，直到没有可翻动
 * 的牌为止。
 *
 * 问题
 *
 * 求当所有牌不再变动时，所有背面朝上的牌的数字。
 *
 * @description 翻牌解题实现
 *              思路：这个题就用一个数组来模拟牌组即可，下标+1对应牌面大小，数组内存储0和1来录牌的状态（0背面1正面）。
 *              循环数组大小次翻牌操作，然后输出数组内值为0的对应下标+1，则为执行完所有操作仍然是背面向上的牌。
 *
 * @version V1.0
 * @Package: cn.dattyrabbit.programerArithmeticTopic.primer.q3.turnTheCardsOver
 * @author: 丁奕
 * @date: 2020-09-07 10:55
 **/
public class TurnTheCardsOver {
    //用于保存卡牌状态的数组。0代表背面，1代表正面，初始化时均为0
    private int cards[];

    /**
     * 有参构造方法，用于创建对象时初始化牌组大小
     * @param size
     */
    public TurnTheCardsOver(int size){
        this.cards = new int[size];
    }

    /**
     * 执行翻牌操作，并最终输出背面朝上的卡牌的数字。
     */
    public void turnOver(){
        //遍历操作，按照规则进行翻牌
        //执行遍历多轮翻牌
```

```
        for(int i = 2; i <= cards.length; i++){
            //执行单轮的翻牌操作，遍历该次需要进行操作的数组内元素，进行异或操作模拟翻牌动作
            for(int j = i-1; j < cards.length; j+=i){
                cards[j] = cards[j]^1;
            }
        }

        //输出打印的前缀
        System.out.print("执行完毕后，仍然背面向上的牌为：");
        //输出背面朝上的卡牌的数字（下标+1）
        for(int i = 0; i < cards.length; i++){
            if(cards[i] == 0){
                System.out.print( i + 1 + " " );
            }
        }

    }

}
```

然后老样子再写一个测试类执行

```
/**
 * 翻牌测试
 *
 * @version V1.0
 * @Package: primer.q3.turnTheCardsOver
 * @author: 丁奕
 * @date: 2020-09-07 11:51
 **/
public class TurnTheCardsOverTest {
    public static void main(String[] args) {
        long start = System.currentTimeMillis();
        TurnTheCardsOver turnTheCardsOver = new TurnTheCardsOver(100);
        turnTheCardsOver.turnOver();
        long end = System.currentTimeMillis();
        System.out.println("总共用时：" + (end-start) +"ms");
    }
}
```

程序执行结果如下

```java
 * @date: 2020-09-07 11:51
**/
public class TurnTheCardsOverTest {
    public static void main(String[] args) {
        long start = System.currentTimeMillis();
        TurnTheCardsOver turnTheCardsOver = new TurnTheCardsOver( size: 100);
        turnTheCardsOver.turnOver();
        long end = System.currentTimeMillis();
        System.out.println("总共用时：" + (end-start) +"ms");
    }
}
```

TurnTheCardsOverTest › main()

Run: TurnTheCardsOverTest

```
"D:\Program Files\Java\jdk1.8.0_221\bin\java.exe" ...
执行完毕后，仍然背面向上的牌为：1 4 9 16 25 36 49 64 81 100 总共用时：1ms

Process finished with exit code 0
```

4: Run 5: Debug 6: TODO Terminal 9: Version Control 0: Messages

Compilation completed successfully in 615 ms (moments ago)

改变参数再来一次

```java
**/
public class TurnTheCardsOverTest {
    public static void main(String[] args) {
        long start = System.currentTimeMillis();
        TurnTheCardsOver turnTheCardsOver = new TurnTheCardsOver( size: 1025);
        turnTheCardsOver.turnOver();
        long end = System.currentTimeMillis();
        System.out.println("总共用时：" + (end-start) +"ms");
    }
}
```

TurnTheCardsOverTest › main()

Run: TurnTheCardsOverTest

```
"D:\Program Files\Java\jdk1.8.0_221\bin\java.exe" ...
执行完毕后，仍然背面向上的牌为：1 4 9 16 25 36 49 64 81 100 121 144 169 196 225 256 289 324 361 400 441 484 529 576 625 676 729 784 841 900 961 1024 总共用时：2ms

Process finished with exit code 0
```

不同思路的对比

这一次跟作者实现的思路没什么太大的差别，不过作者在给出实现代码后，探讨了结果的数字都是平数这一点，然后通过数学方式推导出来了结果还是让我获得了一个新的思路。这不经让我想起了最初代码的时候做的一些题，写代码模拟人去操作之前，如果先多思考下怎么省略不必要的操作，从数学角度给出推导之后再考虑编程，一般都会效率更高。

不过作者原书中给的是ruby代码，我就不贴出代码运行结果了。但是我有一点没明白，就是作者给出第二段所谓的简化后的程序，我虽然没学过ruby，但是看他这个语义也能知道大概意思。为什么明明一段代码就是正常的两层循环，外层是一个O(n)，内层是一个O(logn)，总体是一个O(nlogn)的时间杂度。但是作者第二段代码给它改成了O(n^2)的复杂度，却又说是简化了。实在是没明白其中含义。

所以我就写了一段java的来模拟作者给的这段ruby。然后把牌组改为9999张，运行。

```java
/**
 * 模拟作者的写法，验证得出实际上此写法有问题
 */
public void turnOverByAuthor(){
    for(int i = 1; i <= cards.length; i++){
        boolean flag = false;
        for(int j = 1; j <= cards.length; j++){
            if(i % j == 0){
                flag = !flag;
            }
        }
        if(flag){
            System.out.println(i);
        }
    }
}
```

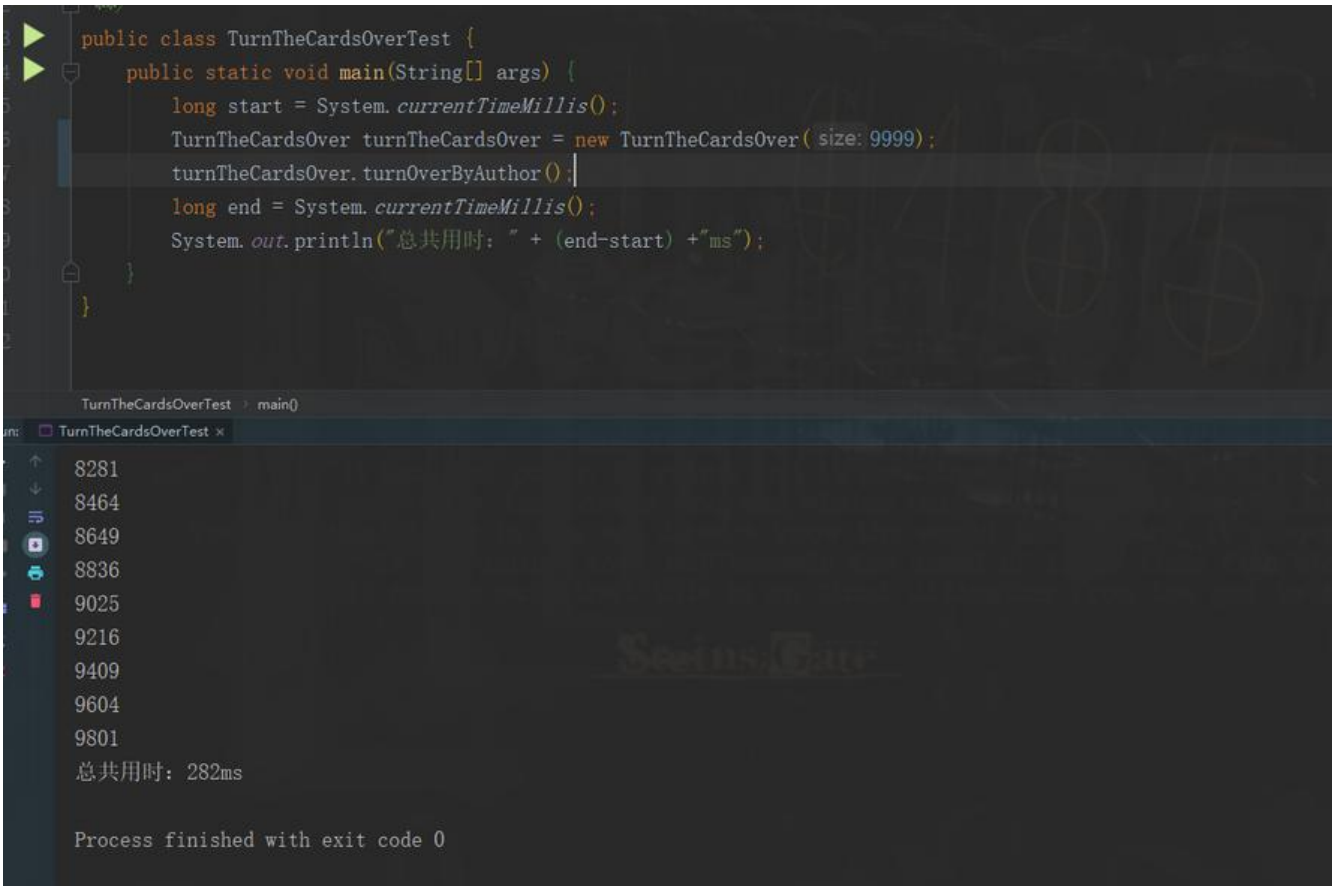
```
public class TurnTheCardsOverTest {
    public static void main(String[] args) {
        long start = System.currentTimeMillis();
        TurnTheCardsOver turnTheCardsOver = new TurnTheCardsOver( size: 9999);
        turnTheCardsOver.turnOver();
        long end = System.currentTimeMillis();
        System.out.println("总共用时：" + (end-start) +"ms");
    }
}

TurnTheCardsOverTest  main()

Run:  TurnTheCardsOverTest
"D:\Program Files\Java\jdk1.8.0_221\bin\java.exe" ...
执行完毕后，仍然背面向上的牌为：1 4 9 16 25 36 49 64 81 100 121 144 169 196 225 256 289 324 361 400 441 484 529 576 625 676 729 784 841 900 961 1024 1089 1156 1225 1
总共用时：6ms

Process finished with exit code 0

4: Run  5: Debug  6: TODO  Terminal  9: Version Control  0: Messages
```



果然是意料之中，所以我个人感觉作者在这里给出的后一段代码可能有点没必要。

总结

这一次的题目相当的简单，而且作者给出的所谓简化代码其实让我挺匪夷所思的。

但是这一篇里面作者跟大家讨论“讨厌麻烦的人比较适合做程序员吗”这个命题，我感觉我也是深有会。

因为当初我本科专业并不是CS相关的，是生化环材之一，毕业之后转行去做了电商的运营（当时是在州的一家专门给各种500强的企业提供专业电商支持服务的，就是替大品牌搞代营的天猫旗舰店）。

因为是在电商的业务一线工作，所以真的如果有做过这一行的朋友，应该对这份工作中经常要做的一机械重复的事情而感到麻烦是深有体会。

当时我就是因为觉得做机械重复的事情太累了，而且做久了人的头真的会晕掉，导致后面频繁的出错才在网上自己学习了按键精灵这个工具，然后学习写一些脚本来模拟我的人工操作。从而代替我来完这些机械重复的工作。这也是后来我转行学习编程的一个契机吧，因为我真的觉得通过编程来处理这机械重复的工作真的很棒很酷很有趣。

不过可能我并不像此书的作者一般，假如要做30分钟的机械重复工作，去花一个小时去做编程。我一会分析下这个机械重复的事情，有没有重用的地方，是不是需要重复操作，如果是很偶然的只做这一就可能再也不会遇到的情况，我大概率就会觉得算了还是手动来吧。但是如果耗时足够长的话，我也定是会毫不迟疑的去考虑代码实现。

当然生活中就算没法用计算机来解决的事情，我也会用一些类似的方法和思路来做。比如之前朋友结让我们5，6个人去他家给他包装喜糖。我就把每个人独立完成整套工序改成了流水线作业，实话实说是极大的提升了干活的效率的。

有一点点扯远了，不过这个话题我确实是有好多东西深有感悟，可能过几天专门写一篇随笔来聊聊我如何转行的经历吧(写完之后再贴链接过来，感觉挺有意思的)。不过今天就到这里了，peace
ray

2020.9.26 23:24

转行经历的文章总算是写完了，贴一下：

我的程序转行之路
