链滴

程序员的算法趣题系列 -Q01- 回文数

作者：DattyRabbit
原文链接：https://ld246.com/article/1597896226063
来源网站：链滴

<h2 id="前言">前言</h2>
此篇为《程序员的算法趣题》中的入门篇第一题"回文数"的相关解题分析博文。
关于该系列的介绍请看：
《程员的算法趣题》-开坑记录
<h2 id="题目">题目</h2>
如果把某个数的各个数字按相反的顺序排列，得到的数和原来的数相同，则这个数就是“回文数。
譬如 123454321 就是一个回文数。
问题
求用十进制、二进制、八进制表示都是回文数的所有数字中，大于十进制数 10 的最小值
例） 9（十进制数） = 1001（二进制数）= 11（八进制数）
※本例中的十进制数 9 小于 10，因此不符合要求。
表 1
<table>
<thead>
<tr>
<th>十进制数</th>
<th>二进制数</th>
<th>八进制数</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
<td>5</td>
</tr>

<tr>
<td>6</td>
<td>110</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>111</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>10</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>12</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>14</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>15</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>16</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>17</td>
</tr>
<tr>
<td>16</td>
<td>10000</td>
<td>20</td>

</tr>
</tbody>
</table>
<img src="https://ld246.com/images/img-loading.svg" alt="问题 HINT.png" data-src="https://b3logfile.com/file/2020/08/%E9%97%AE%E9%A2%98HINT-c4e7dc59.png?imageView2/2/interlace/1/format/jpg">
<h2 id="作者思路及代码实现">作者思路及代码实现</h2>
因为是二进制的回文数，所以如果最低位是 0，那么相应地最高位也是 0。但是，以 0 开头肯定不恰当的，由此可知最低位为 1。
如果用二进制表示时最低位为 1，那这个数一定是奇数，因此只考虑奇数的情况就可以。接下来以简单地编写程序，从 10 的下一个数字 11 开始，按顺序搜索。譬如用 Ruby 就可以通过下面的代找到符合条件的数（代码清单）。
（以下为代码清单 01.01）
<pre><code class="highlight-chroma"># 从11开始搜索
num = 11
while true
    if num.to_s == num.to_s.reverse &&
      num.to_s(8) = num.to_s(8).reverse &&
      num.to_s(2) = num.to_s(2).reverse
      puts num
      break
    end
    #只搜索奇数，次加2
    num += 2
end
</code></pre>
作者的小 TIPS：
<img src="https://ld246.com/images/img-loading.svg" alt="代码 0101tips.png" data-src="https://b3logfile.com/file/2020/08/%E4%BB%A3%E7%A0%810101tips-0e7399c1.png?imageiew2/2/interlace/1/format/jpg">

下面试着用 JavaScript 实现同样的逻辑。JavaScript 里没有内置把字符串逆序的标准函数，因首先需要封装一个返回逆序字符串的方法，其他流程则和代码清单 01.01 中的一致。JavaScript 版本实现如代码清单 01.02 所示。
<pre><code class="highlight-chroma">/* 为字符串类型添加返回逆序字符串的方法 */
String.prototype.reverse = function (){
    return this.split("").reverse().join("");
}

/* 从11 开始搜索 */
var num = 11;
while (true){
    if ((num.toString() == num.toString().reverse()) &&
      (num.toString(8) == num.toString(8).reverse()) &&

```
</span></span><span class="highlight-line"><span class="highlight-cl">        (num.toStrin
(2) == num.toString(2).reverse())){
</span></span><span class="highlight-line"><span class="highlight-cl">        console.log(
um);
</span></span><span class="highlight-line"><span class="highlight-cl">        break;
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">    /* 只搜索奇数
每次加2 */
</span></span><span class="highlight-line"><span class="highlight-cl">    num += 2;
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
<p><img src="https://ld246.com/images/img-loading.svg" alt="代码 0102tips.png" data-src
"https://b3logfile.com/file/2020/08/%E4%BB%A3%E7%A0%810102tips-acef5e1d.png?imageV
ew2/2/interlace/1/format/jpg"></p>
<blockquote>
<p><strong>Point</strong></p>
<p>很多语言都提供了把整数转换成二进制数或者八进制数的方法。表 2 汇总了代表性语言的相关函
或者方法，不过 C 语言并没有提供直接转换的接口。</p>
<p><strong>表 2　各编程语言中进制转换的接口</strong></p>
</blockquote>
<table>
<thead>
<tr>
<th>语言</th>
<th>二进制数</th>
<th>八进制数</th>
<th>十六进制数</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ruby</td>
<td>to_s(2)</td>
<td>to_s(8)</td>
<td>to_s(16)</td>
</tr>
<tr>
<td>PHP</td>
<td>decbin</td>
<td>decoct</td>
<td>dechex</td>
</tr>
<tr>
<td>Python</td>
<td>bin</td>
<td>oct</td>
<td>hex</td>
</tr>
<tr>
<td>JavaScript</td>
<td>toString(2)</td>
<td>toString(8)</td>
<td>toString(16)</td>
</tr>
```

<tr>
<td>Java</td>
<td>toBinaryString</td>
<td>toOctalString</td>
<td>toHexString</td>
</tr>
<tr>
<td>C#</td>
<td>Convert.ToString</td>
<td>Convert.ToString</td>
<td>Convert.ToString 或者 ToString("X")</td>
</tr>
</tbody>
</table>
<img src="https://ld246.com/images/img-loading.svg" alt="答案 tips.png" data-src="https://b3logfile.com/file/2020/08/%E7%AD%94%E6%A1%88tips-83aa8208.png?imageView2/2/interlace/1/format/jpg">
<h2 id="答案">答案</h2>
585（二进制数是 1001001001，八进制数是 1111）
<h2 id="自己做的思路及实现">自己做的思路及实现</h2>
看到题目，首先对比了以下表 1 中的数，便很容易得出一个规律，就是在 1-100 中，3 种不同制表示下，十进制能产生的回文数最少，然后是八进制然后是二进制。（这个应该很容易想明白吧，个位能用的数字越多，碰撞越少）
所以想明白这一点之后，就考虑怎么去查找十进制下大于 10，然后同时满足三种进制表示都为文数的数字。
我自己是这样想的，既然要看是不是回文数，那我就干脆用相对能产生回文数少的十进制表示数构造回文数。比如从 1-9999。我可以用 1-99 来构造回文数。因为题目要满足三种进制表示均为回数，所以我就不去在十进制下一个个往上加着去遍历判断。直接构造出来十进制下的回文数，然后去断八进制表示是不是，再判断二进制表示是不是。
这里解释下为如何在用 1-99 来构建 1-9999 里面的回文数。
可能大多数人不需要解释就能想到，但这里还是解释下。
1-9 就不说了，分别复制一份就是 11，22，33...88，99。
然后是大于 10 的，这里举例 10，21，99。
10 可以构建两个回文数，一个是 1001，一个是 101。
21 同样是两个，2112 和 212。
99 也是两个，9999 和 999。
基本思路就是这样，下面我们来看看代码如何实现吧。
这里我分别写了四个函数，对应不同的功能。
第一个函数是一个通过传入两个参数，来返回满足条件的回文数的入口方法。两个参数作用分别，
<blockquote>
min 符合要求的数不小于 min
buildStart 构造回文开始的数
</blockquote>
该函数代码如下
<pre><code class="highlight-chroma">/*
     * 构造回文，并出符合要求的数
     * @param min 符合要求的数不小于min
     * @param builStart 构造回文开始的数
     * @return

```
</span></span><span class="highlight-line"><span class="highlight-cl">     */
</span></span><span class="highlight-line"><span class="highlight-cl">    public int findPalindromicNumber(Integer min, Integer buildStart){
</span></span><span class="highlight-line"><span class="highlight-cl">        String buildStartStr = buildStart.toString();
</span></span><span class="highlight-line"><span class="highlight-cl">        //初始化符合求的回文数返回值
</span></span><span class="highlight-line"><span class="highlight-cl">        Integer resulNum = null;
</span></span><span class="highlight-line"><span class="highlight-cl">        //是否进行奇构造
</span></span><span class="highlight-line"><span class="highlight-cl">        Boolean odd = true;
</span></span><span class="highlight-line"><span class="highlight-cl">        //遍历中的i长度
</span></span><span class="highlight-line"><span class="highlight-cl">        Integer lengt_i = buildStart.toString().length();
</span></span><span class="highlight-line"><span class="highlight-cl">        //构造数的长

</span></span><span class="highlight-line"><span class="highlight-cl">        Integer lengt_b = buildStartStr.length();
</span></span><span class="highlight-line"><span class="highlight-cl">        for(Integer i  buildStart; length_i &lt;= length_b; i ++){
</span></span><span class="highlight-line"><span class="highlight-cl">            length_i = itoString().length();
</span></span><span class="highlight-line"><span class="highlight-cl">            if(odd == true &amp;&amp; length_i &gt; length_b){
</span></span><span class="highlight-line"><span class="highlight-cl">                //将遍数还原，开始进行偶数位的回文数构造
</span></span><span class="highlight-line"><span class="highlight-cl">                odd = false;
</span></span><span class="highlight-line"><span class="highlight-cl">                i = builStart;
</span></span><span class="highlight-line"><span class="highlight-cl">            }
</span></span><span class="highlight-line"><span class="highlight-cl">            //进行回数构造
</span></span><span class="highlight-line"><span class="highlight-cl">            Integer palndromicNum = buildPalindromic(i, odd);
</span></span><span class="highlight-line"><span class="highlight-cl">            //如果构出的数既大于传入的最小值，又符合八进制二进制是回文数的要求，则返回该值。
</span></span><span class="highlight-line"><span class="highlight-cl">            if(palindrmicNum &gt; min &amp;&amp; checkResult(palindromicNum)){
</span></span><span class="highlight-line"><span class="highlight-cl">                return plindromicNum;
</span></span><span class="highlight-line"><span class="highlight-cl">            }
</span></span><span class="highlight-line"><span class="highlight-cl">        }
</span></span><span class="highlight-line"><span class="highlight-cl">        //遍历结束，未找到结果，从新设置buildStart的值，开始递归
</span></span><span class="highlight-line"><span class="highlight-cl">        buildStart = (nt)Math.pow(10, buildStartStr.length());
</span></span><span class="highlight-line"><span class="highlight-cl">        resultNum = indPalindromicNumber(min, buildStart);
</span></span><span class="highlight-line"><span class="highlight-cl">        return resultum;
```

    }
</code></pre>
因为用一个数来直接构造回文数，当这个数是大于等于 10 的时候，可以构造位数是奇数个的回数和位数是偶数个的回文数（101 和 1001）。
所以在遍历的时候，需要对 遍历数进行按位区分，n 位数(n>=2)的数进行回文数构造，需要进行位数是奇数的回文数构造，如果位数为奇数的回文数构造验证完之后仍未有结果，再从头遍历 n 数，用其构造位数为偶数位的回文数，并验证。
例如，10 -> 101 验证不通过，11 -> 111 验证不通过 .... 99 -> 999(假设此前都没有果)。返回从 10-99 的遍历，这次生成形如 1001,1111,1221 的位数为偶数的回文数，并依次验证。
然后是入口函数中用到的另外两个函数 buildPalindromic 和 checkResult。直接贴代码，注释面有解释函数作用。
<pre><code class="highlight-chroma">/**
     * 传入用作构建文数的数字，以及构建出的回文数位数是奇数偶数为参数，从而构建回文数
     * @param sourceNum 原始数字，用于构建回文数的初始值
     * @param odd 是否构建奇数型回文，默认为true
     * @return
     */
    public int buildPalindromic(Integer sourceNum, Boolean odd){
        //做初始化操，odd参数不传时设置为真
        if(odd == null){
            odd = true;
        }
        if(sourceNum < 1){
            throw new IllegalArgumentException("参数不正确，sourceNum(用于构建的数)不能小于1");
        }
        //开始构建
        String sourceumStr = sourceNum.toString();
        StringBuffer rllbackNumStr = new StringBuffer(sourceNumStr).reverse();
        //初始化要返的回文数StringBuffer型
        StringBufferalindromiceNum = new StringBuffer();
        //先拼接上原的构造数
        palindromiceum.append(sourceNumStr);
        if(odd){
            //构建位数奇数个的回文数
            palindromieNum.append(rollbackNumStr.substring(1, sourceNumStr.length()));
        }else{
</code></pre>

```
</span></span><span class="highlight-line"><span class="highlight-cl">            //构建位数偶数个的回文数
</span></span><span class="highlight-line"><span class="highlight-cl">            palindromieNum.append(rollbackNumStr.substring(0, sourceNumStr.length()));
</span></span><span class="highlight-line"><span class="highlight-cl">        }
</span></span><span class="highlight-line"><span class="highlight-cl">        return Integer valueOf(palindromiceNum.toString());
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span></code></pre>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight-cl">/**
</span></span><span class="highlight-line"><span class="highlight-cl">     * 验证传入数值否满足要求,由于传入的数就是由十进制状态构建的回文数，故不验证十进制
</span></span><span class="highlight-line"><span class="highlight-cl">     * @param checkNum 需要被检查的数值
</span></span><span class="highlight-line"><span class="highlight-cl">     * @return
</span></span><span class="highlight-line"><span class="highlight-cl">     */
</span></span><span class="highlight-line"><span class="highlight-cl">    public boolean heckResult(Integer checkNum){
</span></span><span class="highlight-line"><span class="highlight-cl">        //先转化成八制，然后反转，进行判断。
</span></span><span class="highlight-line"><span class="highlight-cl">        String octalString = Integer.toOctalString(checkNum);
</span></span><span class="highlight-line"><span class="highlight-cl">        String rollbacOctalStr = new StringBuffer(octalString).reverse().toString();
</span></span><span class="highlight-line"><span class="highlight-cl">        //判断的时候判断在八进制状态下是否满足，再判断，因为同样从十进制的1到100，八进制的回文数比二进制少
</span></span><span class="highlight-line"><span class="highlight-cl">        if(rollbackOctlStr.equals(octalString)){
</span></span><span class="highlight-line"><span class="highlight-cl">            //通过八进判断，在进行二进制转换，并判断
</span></span><span class="highlight-line"><span class="highlight-cl">            String binayString = Integer.toBinaryString(checkNum);
</span></span><span class="highlight-line"><span class="highlight-cl">            String rollbckBinaryStr = new StringBuffer(binaryString).reverse().toString();
</span></span><span class="highlight-line"><span class="highlight-cl">            if(rollbackBnaryStr.equals(binaryString)){
</span></span><span class="highlight-line"><span class="highlight-cl">                return true;
</span></span><span class="highlight-line"><span class="highlight-cl">            }
</span></span><span class="highlight-line"><span class="highlight-cl">        }
</span></span><span class="highlight-line"><span class="highlight-cl">        return false;
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span></code></pre>
<p>然后是写一个测试类，调用我自己编写的入口方法。这里先再说明一个函数，是第一次调取入口法时，需要传入一个 buildStart 的参数，所以再介绍一下自己实现的另一个方法，就是根据需要查找回文数允许的最小值，去获得构建大于该值的回文数所需要的构建值。getBuildStart。代码如下</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight-cl">/**
</span></span><span class="highlight-line"><span class="highlight-cl">     * 根据最小值，出一个允许的最小构建回文数的值
</span></span><span class="highlight-line"><span class="highlight-cl">     *
</span></span><span class="highlight-line"><span class="highlight-cl">     * 思路: 若要求后获得的回文数不能小于2001，则2001之后的第一个十进制表示的回文数应该是将2001作为字符串
```

待，从中截取一半，取前半段。
     *   若要求最
获得的回文数不能小于201，则201之后的第一个十进制表示的回文数应该是将201作为字符串看待，
中截取一半，不能完全平分，前半段保留多一个字符，取前半段。
     *   既可以通
字符串操作，也可以如下方法所实现，直接用数字的值进行操作。
     * @param min
     * @return
     */
    public int getBu
ldStart(Integer min) {
        if(min <=
00){
            return 1;
        }
        int length =
in.toString().length();
        return min/(i
t)Math.pow(10,(length/2));
    }
</code></pre>
这里解释下最后一句代码， return min/(int)Math.pow(10,(length/2));
一个简单的例子应该就能明白了，加入传入的 min 是 53128。那么很显然大于 53128 的第一个
文数是 53135。那么用来构建 53135 的数字应该是 531 通过奇数个模式去构建。所以根据中截取一
，不能完全平分，前半段保留多一个字符，取前半段的原则，进行该数字操作就可以实现。
<h2 id="不同思路的对比">不同思路的对比</h2>
这里我给出一下我通过 java 代码实现的作者思路。然后写一个测试类去分别使用自己的思路实
的代码和作者的思路实现的代码去运行。对比结果
以下是通过作者的思路实现的代码
<pre><code class="highlight-chroma"><span class="highlight
cl">/**
     * 作者给出的解
实现
     * @param min
     * @return
     */
    public int findPa
indromicNumber(Integer min){
        //开始遍历的

        int num = 0;
        //进行最开始
遍历数的初始化操作，使之为奇数
        if(min%2 ==
){
            num = min
+ 1;
        }else{
            num = min
+ 2;
        }
        //进入循环，
接对这个遍历的数进行3种进制下是否是回文数的判断。
        while (true){

```
</span></span><span class="highlight-line"><span class="highlight-cl">            String nu
Str = Integer.valueOf(num).toString();
</span></span><span class="highlight-line"><span class="highlight-cl">            String nu
StrRollback = new StringBuffer(numStr).reverse().toString();
</span></span><span class="highlight-line"><span class="highlight-cl">            if(numStr.
quals(numStrRollback)){
</span></span><span class="highlight-line"><span class="highlight-cl">                //此处
用我自己所写的判断八进制和二进制是否是回文数的方法，不影响
</span></span><span class="highlight-line"><span class="highlight-cl">                if(check
esult(num)){
</span></span><span class="highlight-line"><span class="highlight-cl">                    break;
</span></span><span class="highlight-line"><span class="highlight-cl">                }
</span></span><span class="highlight-line"><span class="highlight-cl">            }
</span></span><span class="highlight-line"><span class="highlight-cl">            num += 2;
</span></span><span class="highlight-line"><span class="highlight-cl">        }
</span></span><span class="highlight-line"><span class="highlight-cl">        return num;
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span></code></pre>
<p>以下是测试类的代码</p>
<pre><code class="highlight-chroma"><span class="highlight-line"><span class="highlight
cl">/**
</span></span><span class="highlight-line"><span class="highlight-cl"> * 查找回文数测试
</span></span><span class="highlight-line"><span class="highlight-cl"> *
</span></span><span class="highlight-line"><span class="highlight-cl"> * @version V1.0
</span></span><span class="highlight-line"><span class="highlight-cl"> * @Package: PAC
AGE_NAME
</span></span><span class="highlight-line"><span class="highlight-cl"> * @author: 丁奕
</span></span><span class="highlight-line"><span class="highlight-cl"> * @date: 2019-06
03 20:55
</span></span><span class="highlight-line"><span class="highlight-cl"> **/
</span></span><span class="highlight-line"><span class="highlight-cl">public class Palind
omicNumberTest {
</span></span><span class="highlight-line"><span class="highlight-cl">    //时间格式Form
t
</span></span><span class="highlight-line"><span class="highlight-cl">    private static S
mpleDateFormat simpleDateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">    public static vo
d main(String[] arg){
</span></span><span class="highlight-line"><span class="highlight-cl">        Palindromic
umber palindromicNumber = new PalindromicNumber();
</span></span><span class="highlight-line"><span class="highlight-cl">        //一些单个的
法测试
</span></span><span class="highlight-line"><span class="highlight-cl">//        System.out
println(palindromicNumber.findPalindromicNumber(10, 1));
</span></span><span class="highlight-line"><span class="highlight-cl">//        System.out
println(palindromicNumber.buildPalindromic(5, true));
</span></span><span class="highlight-line"><span class="highlight-cl">//        System.out
println(palindromicNumber.checkResult(10001));
</span></span><span class="highlight-line"><span class="highlight-cl">//        System.out
println(palindromicNumber.getBuildStart(2001));
</span></span><span class="highlight-line"><span class="highlight-cl">        //初始化参数
</span></span><span class="highlight-line"><span class="highlight-cl">        int min = 585
```

```
        //使用自己的方法
        System.out.println("***********使用自己的方法***********");
        start(min, true);

        //使用作者的方法
        System.out.println("\n\n***********使用作者的方法***********");
        start(min, false);
    }

    /**
     * 开始执行主测方法，查找符合要求的回文数并输出。
     * @param min 不能小于某个数
     * @doMyFunc oMyFunc 使用自己的方法
     */
    private static void start(int min, boolean doMyFunc){
        //记录开始时，并输出
        Date startTime = new Date();
        System.out.println("查找回文数开始 " + simpleDateFormat.format(startTime));

        PalindromicNumber palindromicNumber = new PalindromicNumber();

        //开始查找符要求的大于参数min的回文数

        int result = 0;
        if(doMyFunc)

            //使用自己的方法
            int buildStart = palindromicNumber.getBuildStart(min);
            result = palindromicNumber.findPalindromicNumber(min, buildStart);
        }else{
            //使用作者的方法
            result = palindromicNumber.findPalindromicNumber(min);
        }
```

```
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">        System.out.println("得到结果为：" + result + "\n八进制:" + Integer.toOctalString(result) +
</span></span><span class="highlight-line"><span class="highlight-cl">                "\n二进:" + Integer.toBinaryString(result));
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">        //记录结束时，并输出
</span></span><span class="highlight-line"><span class="highlight-cl">        Date endTim = new Date();
</span></span><span class="highlight-line"><span class="highlight-cl">        System.out.println("查找回文数结束 " + simpleDateFormat.format(endTime));
</span></span><span class="highlight-line"><span class="highlight-cl">        //计算用时并出
</span></span><span class="highlight-line"><span class="highlight-cl">        Long costTie = endTime.getTime() - startTime.getTime();
</span></span><span class="highlight-line"><span class="highlight-cl">        System.out.println("查找回文数用时 " + costTime + "毫秒");
</span></span><span class="highlight-line"><span class="highlight-cl">
</span></span><span class="highlight-line"><span class="highlight-cl">    }
</span></span><span class="highlight-line"><span class="highlight-cl">}
</span></span></code></pre>
<p>通过控制测试方法中的 min 参数，运行该测试类，我们来看下 min 取 10 和 min 取 585 的运结果。</p>
<p><img src="https://ld246.com/images/img-loading.svg" alt="min10 结果.png" data-src="ttps://b3logfile.com/file/2020/08/min10%E7%BB%93%E6%9E%9C-2935f957.png?imageView22/interlace/1/format/jpg"></p>
<p>上图为 min 取 10 的运行结果</p>
<p><img src="https://ld246.com/images/img-loading.svg" alt="min585 结果.png" data-src=https://b3logfile.com/file/2020/08/min585%E7%BB%93%E6%9E%9C-c7b8fd87.png?imageView2/2/interlace/1/format/jpg"></p>
<p>这一张是 min 取 585 的运行结果</p>
<p>可以看到在跳过原题的第一个有效答案之后，程序耗时出现了比较大的差异。而形成原因也很清。因为原作给出的方法是从 10 进制的数挨个先遍历判断，所以会判断很多十进制下本身就不是回文的情况。</p>
<p>而我自己的方法，也还可以再做优化，因为我不用去考虑十进制下构造的回文数是偶数的情况，一点是作者给出的想法里面的优化，而我在实现的时候却未曾想到这一点。</p>
<h2 id="总结">总结</h2>
<p>从分析对比代码运行结果，就可以知道，所谓算法去优化性能，其实就是在编写代码的时候，去为的先考虑好各种情况，然后在代码层面就去规避一些不必要的判断，就可以达到一定程度的代码执效率的优化。</p>
<p>但是这又带来的另一个问题，就是通过人为的去规避的话，最后到底层的代码实现，可能会出现码量更大，逻辑比简单粗暴的代码实现更难理顺的现象（因为这个代码是在去年写的，现在来写博客要去看当初自己的实现思路，代码为何这样写），就一定程度的提高了维护难度。</p>
<p>所以！注释真的很重要。辛亏当初自己在这部分代码的时候，尽可能的去把注释写得详尽，每个法用来做什么，为啥要这样操作都尽力去写下来。所以时隔一年再回头来看以前的代码，也能很快的顺思路。</p>
<p>永远不要想着当下写的代码当下明白就行。虽然有个梗叫做，这段代码只有上帝和当时写代码的己知道其中的含义。但是开开玩笑就好了，千万别去做这样的事啊。:pray:</p>
```