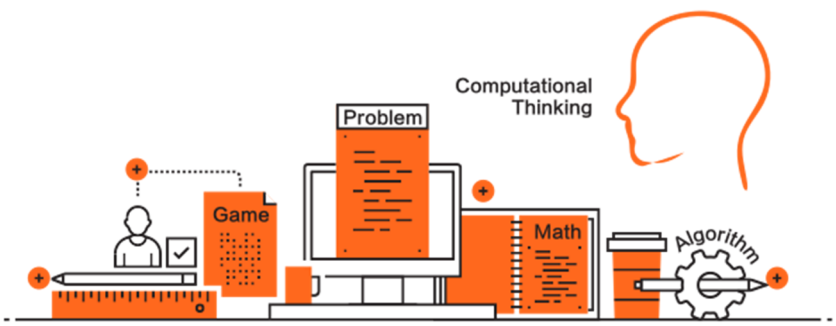
问题 3

奇妙的整除性



游戏

有一个很奇特的数字 4753869120。它的奇特之处在于：

① 恰好是由 0，1，…，9 且每个数字只出现一次构成的 10 位数。

② 都能被 2，3，4，5，6，7，8，9，10，11，12，13，14，15，16，17，18 这连续 17 个数整除！

对于①，可以通过观察得到；对于②，可以通过一个个做除法来检验。

你想不想知道，还有没有别的数也这样奇妙呢？或者反过来问，有什么办法可以确认满足条件①和②的是不是只有这一个数呢？

算法和程序

为了下面的讨论不至太啰唆，我们称前面的特点①为“完整性”、②为“整除性”。这种用相对简洁且比较形象的词语来表达某句话或一段话准确描述的事物，在计算机科学中是常见的。值得注意的是，这类简洁词语所表达的意思通常只是在一定的背景下才适用，换一个场合它们可能就是完全不同的意思。我们这里的“完整性”和“整除性”特指前面①和②描述的某些数可能具有的特性。

我们现在来探究同时满足完整性和整除性的数有哪些，下面有三种方法。

第一种方法

第一种方法，最不费脑筋，不妨称为“蛮力法”。由完整性可知，那样的数一定不会超过 10 位，也就是 10 亿（10^{10}）。我们编个程序一个个看，把满足完整性和整除性的找出来不就可以了？

如何确定一个整数 n 是否满足整除性？也就是检查它是不是能被 2、3、…、18 分别整除，用一个列表 L 将它们放好，用循环结构一个个取出来（for m in L:）测试（if n % m != 0:）即可。

如何确定一个整数 n 是否满足完整性？我们需要一个个取出它的十进制数字，

看看 0、1、2、…、9 是不是都齐了，而且恰好各出现一次。不难想到，n%10 是 n 除以 10 的余数，也就是 n 的个位数字，n//10 则是 n 除以 10 的商。也就是说

```
digit, n = n % 10, n // 10
```

这样就将 n 的个位数字提取出来放到 digit 中，同时 n 自己失去了个位数字，先前的十位数字成了新的个位数字。如此反复，就把构成 n 的各数字一一剥离了。这正是我们需要的。剩下的就是要看怎么控制这个过程的结束，以及保留每次得到的数字，看看 0～9 是不是恰好各出现了一次。

过程的结束，可以通过判断 n 是不是已经变成 0 了。

可以用一个小技巧来帮助判断数字出现的情况，在保留每次得到的数字的同时，也为后面的判断做准备。注意到我们检测的数的范围不超过 10 位数，于是可以说“出现了 10 个不同的数字，当且仅当 0～9 每个数字各出现了一次”。利用 Python 的集合操作功能，每提取一个数字，就把它添加到一个集合中，最后看集合中是否恰好为 10 个元素。

下面就是这样一个程序。

```
L = [2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18]
for n in range(1, 9999999999):                  # 100 亿个数
    dividable = True
    for m in L:                                 # 检查整除性
        if n % m != 0:                          # 如果不成，就跳过剩下的，去测下一个数
            dividable = False
            break
    if dividable:                               # 准备检查完整性
        num = n                                 # n 留着后面还要用，下面用 num 操作
        digits = set()                          # 利用集合的自动去重功能
        while num > 0:                          # 从低位到高位，一个个取出数字
            digits, num = digits|{num%10}, num//10
        if len(digits) == 10:                   # 是不是刚好 10 个数字凑齐了
            print('这个数同时满足两个性质：', n)
```

前面说了，这是一个“蛮力”程序，算法没什么讲究，编程稍用了点技巧。在一台普通笔记本电脑上运行它，大概要花 1 个小时，就挑出了所有同时满足两

个性质的数。不管怎样，它展示了一种可能，也让我们看到了现代计算机的算力——处理 100 亿个数，不是不可以想象的。

当然，我们希望更有效率。

第二种方法

上面的方法是不管三七二十一，将所有 10 位数一个个都看一遍。一个新的想法是，我们只考虑那些已经满足了完整性的数如何？也就是说，那些由 0、1、…、9 的全排列组成的数，于是只需检查它们是否满足整除性就可以了。一共是 $10!=3628800$，即 300 多万，比 100 亿要小多了。

不过，这需要有个办法来生成所有全排列组成的数，本身是一个有些难度的算法，不适合初学者。

第三种方法

那么，有没有可能只考虑那些在 10 位数的范围内已经满足了整除性的数呢？对于它们，我们只需要检查是否满足完整性。这听起来似乎更难。怎么能确保所有满足整除性的数都得到检查呢？下面的分析将给我们信心。

我们关心的是能被 2、3、4、…、17、18 整除的 10 位整数。回顾一下算术基本定理：大于 1 的数能够唯一表示为质数幂次的乘积。一个整数 n 能被那 17 个数整除，当且仅当 n 的质数幂次乘积表示包含它们的质因子。而将它们按照质因子写出来，就是

$$2,3,2^2,5,2\times3,7,2^3,3^2,2\times5,11,2^2\times3,13,2\times7,3\times5,2^4,17,2\times3^2$$

因为高幂次的蕴含着低幂次，去掉那些低幂次的项，就剩下：

$$5,7,3^2,11,13,2^4,17$$

也就是说，如果一个整数 n 包含这些因子，它就具有我们关心的整除性。反之亦然，满足整除性一定要包含这些因子。而这些因子乘起来的结果是 12252240。

这意味着，我们需要考虑的整数是 $12252240\times k$，其中 k 是整数。这样的整数具有整除性，具有整除性的整数一定可以表示为这种形式。

最后注意到，当 $k=817$ 时，$12252240\times k$ 就超过 10 位数了。也就是说，我们最多只需要检查 816 个数。

程序如下：

```
for k in range(1, 817):
    num = 12252240*k                                    # 满足整除性，但不一定恰由 0～9 构成
    digits = set()
    while num > 0:
        digits, num = digits|{num % 10}, num // 10
    if len(digits) == 10:
        print('这个数满足整除性且恰好由 0～9 构成：', 12252240*k)
```

它是不是既高效又简单？

进一步思考

如果运行了上面的程序（无论哪一个），运行结果是什么呢？那种奇特的性质（完整性、整除性）只有 4753869120 满足吗？

一般，“缩小搜索空间”是算法设计的基本追求之一，也就是关于效率追求的基本抓手。上面讨论的问题比较夸张地呈现出了这样一种观念。

一开始，我们只是用了 10 位数的限制，搜索空间的大小是 10^{10}，对其中的每个元素原则上要既检查完整性，也检查整除性。编写程序直截了当，基本不用费脑子，勉强能解决问题。

后来考虑可以将空间限制到所有满足完整性的数，也就是 10 位数的有效排列组合，缩小很多。但是如何列举出该空间的元素本身并不简单。当然，程序设计语言提供的一些内置模块有助于简化编程过程，如 Python 的 itertools 模块提供的排列函数 permutations()可以返回给定数字序列的所有排列组合。注意，permutations()函数的输入和输出都是字符串形式，因此要用 str()和 int()做相应的转换。

最后，我们从数的质因子分解的特点入手，经过分析发现，在 10 位数范围内满足整除性的只有 800 多个，这使得搜索空间极大缩小，而且它们容易被列举出来（$12252240\times k$，$k=1,2,3,\cdots,816$），从而不仅程序运行效率极大提高，编程也十分简单容易。能得到这种“双赢”结果是特别令人高兴的。当然，基础就是进行适当的数学分析。

最后，走过了这个旅程，你也许会好奇，那样的 10 位数居然能被 2、3、4、5、6、7、8、9、10、11、12、13、14、15、16、17、18 这些数整除。再加一个 19 如何？也就是说，是否存在具有我们所说的完整性，且能被 2、3、…、18、19 整除的数呢？相信你很快能找到答案。