Java库谜题59：什么是差 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/138/2021_2022_Java_E5_BA_93_E8_B0_9C_c97_138642.htm 下面的程序在计算一个int数组中的元素两两之间的差，将这些差置于一个集合中，然后打印该集合的尺寸大小。那么，这个程序将打印出什么呢？ import java.util.*. public class Differences { public static void main(String[ ] args) { int vals[ ] = { 789, 678, 567, 456, 345, 234, 123, 012 }. Set diffs = new HashSet(). for (int i = 0. i for (int j = i. j diffs.add(vals[i] - vals[j]). System.out.println(diffs.size()). } } 外层循环迭代数组中的每一个元素，而内层循环从外层循环当前迭代到的元素开始迭代到数组中的最后一个元素。因此，这个嵌套的循环将遍历数组中每一种可能的两两组合。（元素可以与其自身组成一对。）这个嵌套循环中的每一次迭代都计算了一对元素之间的差（总是正的），并将这个差存储到了集合中，集合是可以消除重复元素的。因此，本谜题就带来了一个问题，在由vals数组中的元素结成的对中，有多少唯一的正的差存在呢？ 当你仔细观察程序中的数组时，会发现其构成模式非常明显：连续两个元素之间的差总是111。因此，两个元素之间的差是它们在数组之间的偏移量之差的函数。如果两个元素是相同的，那么它们的差就是0；如果两个元素是相邻的，那么它们的差就是111；如果两个元素被另一个元素分割开了，那么它们的差就是222；以此类推。看起来不同的差的数量与元素间不同的距离的数量是相等的，也就是等于数组的尺寸，即8。如果你运行该程序，就会发现它打印的是14。怎么回事呢？ 上面的分析有一个小的漏洞。要想了

解清楚这个缺陷，我们可以通过将println语句中的.size()这几个字符移除掉，来打印出集合中的内容。这么做会产生下面的输出： [111,222,446,557,668,113,335,444,779,224,0,333,555,666] 这些数字并非都是111的倍数。在vals数组中肯定有两个毗邻的元素的差是113。如果你观察该数组的声明，不可能很清楚地发现原因所在： int vals[ ] = { 789, 678, 567, 456, 345, 234, 123, 012 }. 但是如果你打印数组的内容，你就会看见下面的内容：[789,678,567,456,345,234,123,10] 为什么数组中的最后一个元素是10而不是12呢？因为以0开头的整数类型字面常量将被解释成为八进制数值[JLS 3.10.1]。这个隐晦的结构是从C编程语言那里遗留下来东西，C语言产生于1970年代，那时八进制比现在要通用得多。 一旦你知道了012 == 10，就会很清楚为什么该程序打印出了14：有6个不涉及最后一个元素的唯一的非0差，有7个涉及最后一个元素的非0差，还有0，加在一起正好是14个唯一的差。订正该程序的方法更加明显：将八进制整型字面常量012替换为十进制整型字面常量12。如果你这么做了，该程序将打印出我们所期望的8。 本谜题的教训很简单：千万不要在一个整型字面常量的前面加上一个0；这会使它变成一个八进制字面常量。有意识地使用八进制整型字面常量的情况相当少见，你应该对所有的这种特殊用法增加注释。对语言设计者来说，在决定应该包含什么特性时，应该考虑到其限制条件。当有所迟疑时，应该将它剔除在外。