<https://10.1pxeye.com/negative-remainder-operation/>

取模运算（“Modulo Operation”）和取余运算（“Complementation ”）两个概念有重叠的部分但又不完全一致。主要的区别在于对负整数进行除法运算时操作不同。

取模主要是用于计算机术语中。取余则更多是数学概念。

**讨论：**

首先，看看自然数的取模运算（**定义1**）:

如果 **a** 和 **b** 是两个自然数，**b** 非零，可以证明存在两个唯一的整数 **c** 和 **r**，满足 **a** = **cb** + **r** 且0 ≤ **r** < **b**。其中，**c** 被称为商，**r** 被称为余数。

假如我们按照正数求余的规则求 (-7) mod 3 的结果，就可以表示 -7 为 (-3)\* 3 +2。其中，2是余数，-3是商。

下面是几种软件中对此的理解。

|  |  |  |
| --- | --- | --- |
| 语言 | 语句 | 输出 |
| C++（G++ 编译） | cout << (-7) % 3; | -1 |
| Java（1.6） | System.out.println((-7) % 3); | -1 |
| Python 2.6 | (-7) % 3 | 2 |
| [百度计算器](http://www.baidu.com/s?bs=(-7)+mod+2&f=8&wd=(-7)+mod+3) | (-7) mod 3 | 2 |
| [Google 计算器](http://www.google.com.hk/search?sourceid=chrome&ie=UTF-8&q=(-7)+mod+3) | (-7) mod 3 | 2 |

可以看到，答案不是唯一。所以不能直接把正数的法则用在负数的取余上。

实际上，在**整数范围**内，自然数的求余法则并不被很多人所接受，大家大多认可的是下面的这个**定义2**。

如果**a** 与**b** 是**整数**，**b** 非零，那么**余数** **r** 满足这样的关系：a = cb + r , c 为整数，且0 ≤ |r| < |b|。

可以看到，这个定义导致了有负数的求余并不是我们想象的那么简单，比如，-1 和 2 都是 (-7) mod 3 正确的结果，因为这两个数都符合定义。这种情况下，对于取模运算，可能有两个数都可以符合要求。我们把 -1 和 2 分别叫做**正余数**和**负余数**。

通常，当除以d时，如果正余数为r1，负余数为r2，那么有：

r1 = r2 + d

对负数余数不明确的定义可能导致严重的计算问题，对于处理关键任务的系统，错误的选择会导致严重的后果。

看完了 (-7) mod 3，下面我们来看一看 7 mod (-3) 的情况（看清楚，上面的例子 7 带负号，现在是 3 带负号）。

根据**定义2**，7 = (-3) \* (-2) + 1 或7 = (-3) \* (-3) -2，所以余数为 1 或 -2。

|  |  |  |
| --- | --- | --- |
| 语言 | 语句 | 输出 |
| C++（G++ 编译） | cout << 7 % (-3); | 1 |
| Java（1.6） | System.out.println(7 % (-3)); | 1 |
| Python 2.6 | 7 % (-3) | -2 |
| [百度计算器](http://www.baidu.com/s?bs=(-7)+mod+2&f=8&wd=(-7)+mod+3) | 7 mod (-3) | -2 |
| [Google 计算器](http://www.google.com.hk/search?sourceid=chrome&ie=UTF-8&q=(-7)+mod+3) | 7 mod (-3) | -2 |

从中我们看到：

Java 紧随 C++ 的步伐，而 Python、Google、百度步调一致。难道真是**物以类聚？**

可以推断，C++ 和 Java 通常会**尽量让商更大一些尽量让商更小是在正整数运算中，所有语言和计算器都遵循了尽量让商小的原则，**

如果按照第二点的推断，我们测试一下 (-7) mod (-3)，结果应该是前一组语言（C++，Java）返回 2，后一组返回 -1。（请注意这只是假设）

于是有了如下实际测试：

|  |  |  |
| --- | --- | --- |
| 语言 | 语句 | 输出 |
| C++（G++ 编译） | cout << -7 % (-3); | -1 |
| Java（1.6） | System.out.println(-7 % (-3)); | -1 |
| Python 2.6 | -7 % (-3) | -1 |
| [百度计算器](http://www.baidu.com/s?bs=(-7)+mod+2&f=8&wd=(-7)+mod+3) | -7 mod (-3) | -1 |
| [Google 计算器](http://www.google.com.hk/search?sourceid=chrome&ie=UTF-8&q=(-7)+mod+3) | -7 mod (-3) | -1 |

**结果，所有语言和计算机返回结果完全一致。**

**小结：**

我们由此可以总结出下面两个结论：

对于任何**同号使商尽可能小**

对于异号的两个整数，C++/Java语言的原则是**使商尽可能大，使商尽可能小。**

**总结：**

假设求余数（模）r，c是商：

求余方法a%b，求模方法a mod b，则余数r为：

计算模或者余数： r = a – c\*b

那么，求模运算和求余运算在第一步不同: 取余运算在取c的值时，向 0 方向舍入(fix()函数)；

而取模运算在计算c的值时，向负无穷方向舍入(floor()函数)。

例如：计算-7 Mod 4

那么：a = -7；b = 4；

第一步：求整数商c，如进行求模运算c = -2（向负无穷方向舍入），求余c = -1（向0方向舍入）；

第二步：计算模和余数的公式相同，但因c的值不同，求模时r = 1，求余时r = -3。

**归纳：**

当a和b符号一致时，求模运算和求余运算所得的c的值一致，因此结果一致。

当符号不一致时，结果不一样。

求模运算结果的符号和b一致，求余运算结果的符号和a一致。

另外各个环境下%运算符的含义不同，比如c/c++，java 为取余，而python则为取模。

<https://www.cnblogs.com/m2364532/p/16901656.html>

**数学定义上的取余: (余数必须大于0)**

如果a和d是两个自然数，d非零，可以证明存在两个唯一的整数 q 和 r，满足a=qd+r且0 ≤ r < d（其中q为商，r为余数）。

举例：

5%3=3x1+2，商为1，余数为2

(-5)%(-3)=(-3)x2+1，商为2，余数为1

5%(-3)=(-3)x(-1)+2，商为-1，余数为2

(-5)%3=3x(-2)+1，商为-2，余数为1

**计算机中取余：**

所有语言和计算器都遵循了尽量让**商尽量靠近0**的原则，即5%(-3) 的结果为2而不是-1，(-5)%3的结果是-2而不是1。

如：-5%3=-2, -5%-3=-2, 5%-3=2

即:**都先看成两个都是正数，然后取余，如果a是正数，结果就是正数，如果a是负数，结果就是负数，b的正负不影响。**

**编程中考虑负数取余的情况（化成数学上的余数）**

**(a % b + abs(b)) % b**

**典型的例子是**[L974.Subarray Sums Divisible by K](note://WEBde138f801994c4eb8fff2da131b5856f)