巧用数学 AC 算法面试题

原创：Alpinist Wang LeetCode力扣 7月18日

计算机的基础是数学，所以数学功底对编程也很重要。本篇将讲述常见的数学问题，主要做法是运用一些数学上的小技巧，包括常见的数字操作与位运算。另外，一些算法题目可以用数学计算出通项公式，从而避免使用算法计算，无论从时间复杂度和空间复杂度来看，都比使用算法好。

获取数字的每一位

第一种做法：通过 `%` 求余运算获取数字的最后一位，通过整数除法 `/10` 去掉最后一位，重复此操作直至数字变为 0，便可获取到数字的每一位。

示例：7. 整数反转

力扣的这道题目非常简洁，看完立刻能够明白题意。做法也很明确，只要将数字的每一位取出，再倒置即可。

文中代码部分可将屏幕设置为 自动旋转 横屏观看

使用以上技巧，我们可以写出如下代码：

```
fun reverse(x: Int): Int {
    var num = x.toLong()
    var ans = 0L
    while (num != 0L) {
        // 乘以10将个位数空余出来
        // 使用%对num求余，获取最后一位，加到ans的个位上
        ans = ans * 10 + num % 10
        // 通过整数除法/10去掉最后一位
        num /= 10
    }
    // 根据题意，如果越界返回0
    if (ans > Int.MAX_VALUE || ans < Int.MIN_VALUE) ans = 0
    return ans.toInt()
}
```

这种做法的好处是对正数和负数都可以采用一致的处理方式。

除此之外，还有另一种做法。

第二种做法：将数字转成 String，遍历每一位字符，使用 `char - '0'` 将字符变为数字，重复此操作直至字符串为空，即可获取到每一位数字。

采用这种做法，代码如下：

```
class Solution {
    fun reverse(x: Int): Int {
        var num = x.toString()
        var ans = 0L
        // 如果有符号位，记录符号位
        var flag = true
        if (num[0] == '+') {
            num = num.drop(1)
        }
```

```
        if (num[0] == '-') {
            flag = false
            num = num.drop(1)
        }
        while (num.isNotEmpty()) {
            // 将数字转成String，遍历每一位字符，使用char - '0'将字符变为数
            ans = ans * 10 + (num.last() - '0')
            if (ans > Int.MAX_VALUE || ans < Int.MIN_VALUE) ans = 0
            num = num.dropLast(1)
        }
        if (!flag) ans = -ans
        return ans.toInt()
    }
}
```

采用字符串获取数字每一位时，需要单独处理符号位。

应用：将字符串转换成数字(atoi 问题)

atoi（ASCII to int）问题在面试中很常见，虽然问题看起来很简单，但有很多细节需要注意，稍不留神就容易出错。所以比较考验开发者写程序的仔细程度与考虑问题的全面性。

对于这类问题，需要考虑以下三点：

- 1.结果的范围：是否包含正数、负数、0
- 2.字符串里面是否包含非数字字符，对于非数字字符怎么处理。
- 3.是否会超出 Int 范围或者 Long 范围。

力扣 atoi 典型题：8. 字符串转换整数 (atoi)

此题只需要使用以上数学技巧，并按照题目描述写出程序即可。我们分以下几步解决：

- 1.根据题目描述，将前缀的空格去掉；
- 2.判断正负号，并记录；
- 3.遍历字符串，扫描到非数字字符则不再继续遍历；
- 4.判断数字是否越界；

- 5.根据符号位调整结果的正负。

代码如下：

```
fun myAtoi(str: String): Int {
    // 起始下标
    var index = 0
    // 根据题意，去掉前缀空格
    while (index < str.length && str[index] == ' ') index++
    // 判断符号位，是否是负数
    var flag = false
    if (index < str.length && str[index] == '+') {
        index++
    } else if (index < str.length && str[index] == '-') {
        flag = true
        index++
    }
    var result = 0L
    for (i in index until str.length) {
        // 不是数字，跳出循环
        if (str[i] !in '0'..'9') break
        // 后缀此数字
        result = result * 10 + (str[i] - '0')
        // 是否越界
        if (!flag && result > Int.MAX_VALUE) return Int.MAX_VALUE
        if (flag && -result < Int.MIN_VALUE) return Int.MIN_VALUE
    }
    // 根据符号位调整结果的正负
    return if (flag) -result.toInt() else result.toInt()
}
```

位运算

众所周知，计算机内部存储数字使用的是二进制，位运算也是所有运算当中最快的，掌握一些位运算技巧可以提升算法效率。此技巧的原理在于：

```
1.如果 n 以 1 结尾：
xxxxx1
则 n - 1 的二进制表示为：
xxxxx0
此时：n and n - 1 = xxxxx0

2.如果 n 以 0 结尾：
xxx10..0
则 n - 1 的二进制表示为：
xxx01..1
此时：n and n - 1 = xxx00..0

两种情况都去掉了最后一个 1
```

应用一：191. 位 1 的个数

本题只要不断使用 n and n - 1 去掉二进制中最后一个 1，直至数字变为 0 即可。解法如下：

```
fun hammingWeight(n: Int): Int {
    var num = n
    var count = 0
    while (num != 0) {
        num = num and num - 1
        count++
    }
    return count
}
```

应用二：231. 2 的幂

此题的常规做法是：将传入的数字不断地除以 2，如果一直能整除到 1，则此数是 2 的幂。

常规做法的代码如下：

```
fun isPowerOfTwo(n: Int): Boolean {
    return isPower(n, 2)
}

/**
 * 判断n是否是base的幂
 */
fun isPower(n: Int, base: Int): Boolean {
    if (n <= 0) return false
    var num = n
    while (num != 1) {
        // 如果无法整除，则返回false
        if (num % base != 0) return false
        num /= base
    }
    // 直到除到1，n都能base整除，则n是base的幂
    return true
}
```

偷偷告诉你，这一组代码还可以直接拿去解决 326. 3 的幂、342. 4 的幂，只要将传入的 2 改为 3、4 即可。

那么力扣为什么要出三道"一模一样"的题目呢？其实不然，**这三个题的考察点各不相同，需要程序员对数字在计算机中的存储形式有所了解。**

在计算机中，11111 表示 1*2^4+1*2^3+1*2^2+1*2^1+1*2^0，所以 2 的 n 次幂在计算机中存储形式为 1 后面接 n 个 0。例如：

```
1  二进制存储形式 0001
2  二进制存储形式 0010
4  二进制存储形式 0100
```

```
8 二进制存储形式 1000
```

可以看出，如果一个数字去掉二进制的最后一位 1 之后等于 0，则此数就是 2 的幂。使用以上技巧，我们可以一步写出答案：

```
fun isPowerOfTwo(n: Int): Boolean {
    return n > 0 && n and n - 1 == 0
}
```

异或

异或的运算规则如下：~~如果 a 和 b 不同，则异或结果为 1~~，否则异或结果为 0。即：

```
1 xor 0 == 1
0 xor 0 == 0
1 xor 1 == 0
```

相同的数字由于其二进制每一位都相同，异或结果是每一位都为 0，所以有相同的数字异或后为 0。

由于 0 异或 0 等于 0，0 异或 1 等于 1，即：任何数异或 0 都等于其本身。

应用：136. 只出现一次的数字

这是一道 BAT 面试题，使用以上技巧，只需将每一位数字相互异或，出现偶数次的数字会异或成 0，出现奇数次的那个数字会被保留在结果中。解法如下：

```
fun singleNumber(nums: IntArray): Int {
    return nums.reduce { a, b ->
        a xor b
    }
}
```

运用幂的性质

在对数字做因式分解时，分解到质数便不可再分，而质数的幂等于多个质数相乘，所以质数的幂只有质数这一个质因子。

应用：326. 3 的幂

由于 3 的幂只有一个质因子 3。所以我们可以用 3 的高次幂对 n 求余，如果余数为 0，则 n 是 3 的幂。由于 int 的数据范围是[-2^31，2^31-1]，计算可知 int 范围内 3 的最高次幂为 3^19。于是我们有如下解法：

```
fun isPowerOfThree(n: Int): Boolean {
    return n > 0 && Math.pow(3.0, 19.0).toInt() % n == 0
}
```

231. 2 的幂 这一题也可以应用这一性质，在 int 范围内 2 的最高次幂为 2^30，所以代码如下：

```
fun isPowerOfTwo(n: Int): Boolean {
    return n > 0 && Math.pow(2.0, 30.0).toInt() % n == 0
}
```

但是要注意，342. 4 的幂 无法使用这种解法，原因在于 4 不是一个质数。

那么我们顺便来看一下 4 的幂这一题，先列举几个 4 的幂看一下规律：

```
1  0000001
4  0000100
16 0010000
64 1000000
```

可以看出：4 的幂的二进制表示为 1 后面接偶数个 0，即：4 的幂在计算机中的二进制表示只有一个 1，且数字 1 在奇数位置上。

怎么表示这一种关系呢？我们可以在 2 的幂基础上考虑，首先 4 的幂肯定也必须满足 2 的幂条件，即：n and n - 1 == 0，除此之外，我们需要筛选出 2 的幂中，1 在奇数位置上的数字。

考虑一下：二进制中，0010 和 0100 有什么区别？

由于 0100 的奇数位置有 1，0100 的奇数位置没有 1。如果有一个数字的奇数位为 1，如 0100，将它与 0100 和 0100 做与运算，奇数位置没有 1的 0010 将变成 0，奇数位置有 1 的 0100 不变。这个区别就可以作为切入点。

我们用一个奇数位全为 1 的数字：

01010101010101010101010101010101

与 2 的幂做与运算。结果不等于 0 的就是奇数位置有 1 的数字。用 16 进制表示：

01010101010101010101010101010101

即 0x55555555，所以我们有如下代码：

```
fun isPowerOfFour(n: Int): Boolean {
    return n > 0 && n and n - 1 == 0 && (n and 0x55555555 != 0)
}
```

当然，我们也可以用偶数位全为 1 的数字：

10101010101010101010101010101010

与 2 的幂做与运算，结果等于 0 的就是奇数位置有 1 的数字，用 16 进制表示：

10101010101010101010101010101010

即 0xAAAAAAAA，代码如下：

```
fun isPowerOfFour(n: Int): Boolean {
    return n > 0 && n and n - 1 == 0 && (n and 0xAAAAAAAA.toInt() ==
```

```
}
```

巴什博奕

巴什博奕是一种两个人玩的回合制数学战略游戏。游戏者轮流从一堆棋子（或者任何道具）中取走一个或者多个，先取光者胜出。

假设这一堆有 n 个物品，规定每次可以取 1~m 个。我们可以分析出如下规律：

> 如果n = m+1，无论先取的人取走多少个，后取的人都能取光剩余物品，后取者胜。
>
> 如果n = (m+1)*2，无论先取的人取走多少个，后取的人都能将物品取到(m+1)个，后取者胜。
>
> 如果n = (m+1)*3，无论先取的人取走多少个，后取的人都能将物品取到(m+1)*2个，后取者胜。
>
> 如果n = (m+1)*4，无论先取的人取走多少个，后取的人都能将物品取到(m+1)*3个，后取者胜。
>
> ……

> 如果n=c (c<=m)，先取的人可以取光所有物品，先取者胜
>
> 如果n=m+1+c (0<c<=m)，先取的人可以将物品取到(m+1)个，先取者胜
>
> 如果n=(m+1)*2+c (0<c<=m)，先取的人可以将物品取到(m+1)*2 个，先取者胜
>
> 如果n=(m+1)*3+c (0<c<=m)，先取的人可以将物品取到(m+1)*3 个，先取者胜
>
> 如果n=(m+1)*4+c (0<c<=m)，先取的人可以将物品取到(m+1)*4 个，先取者胜
>
> ……

总结可知，如果 n=(m+1)*k，则后取者胜，如果 n=(m+1)*k+c(0<c<=m)，则先取者胜。

应用：292. Nim 游戏

根据巴什博奕的规则，可知如果 n 是 4 的倍数，则后手获胜，否则先手获胜。代码如下：

```
fun canWinNim(n: Int): Boolean {
    return n % 4 != 0
}
```

这样的数学技巧还有很多，数学问题没有固定套路，又全是套路。如果你恰好知道的话，代码将非常简洁。学习方法是平时注意多积累，将遇到的技巧记录下来，使用时就会得心应手。

互动话题

文中提到的数学技巧你有用过嘛？你是否还有更好的解法呢？不妨在留言区与大家分享~

本文作者：Alpinist Wang

编辑&版式：霍霍

推荐阅读