# Leetcode 题解 - 位运算

* [Leetcode 题解 - 位运算](#leetcode-题解---位运算)
	+ [0. 原理](#0-原理)
	+ [1. 统计两个数的二进制表示有多少位不同](#1-统计两个数的二进制表示有多少位不同)
	+ [2. 数组中唯一一个不重复的元素](#2-数组中唯一一个不重复的元素)
	+ [3. 找出数组中缺失的那个数](#3-找出数组中缺失的那个数)
	+ [4. 数组中不重复的两个元素](#4-数组中不重复的两个元素)
	+ [5. 翻转一个数的比特位](#5-翻转一个数的比特位)
	+ [6. 不用额外变量交换两个整数](#6-不用额外变量交换两个整数)
	+ [7. 判断一个数是不是 2 的 n 次方](#7-判断一个数是不是-2-的-n-次方)
	+ [8. 判断一个数是不是 4 的 n 次方](#8--判断一个数是不是-4-的-n-次方)
	+ [9. 判断一个数的位级表示是否不会出现连续的 0 和 1](#9-判断一个数的位级表示是否不会出现连续的-0-和-1)
	+ [10. 求一个数的补码](#10-求一个数的补码)
	+ [11. 实现整数的加法](#11-实现整数的加法)
	+ [12. 字符串数组最大乘积](#12-字符串数组最大乘积)
	+ [13. 统计从 0 ~ n 每个数的二进制表示中 1 的个数](#13-统计从-0-~-n-每个数的二进制表示中-1-的个数)

## 0. 原理

**基本原理**

0s 表示一串 0，1s 表示一串 1。

x ^ 0s = x x & 0s = 0 x | 0s = x
x ^ 1s = ~x x & 1s = x x | 1s = 1s
x ^ x = 0 x & x = x x | x = x

利用 x ^ 1s = ~x 的特点，可以将一个数的位级表示翻转；利用 x ^ x = 0 的特点，可以将三个数中重复的两个数去除，只留下另一个数。

1^1^2 = 2

利用 x & 0s = 0 和 x & 1s = x 的特点，可以实现掩码操作。一个数 num 与 mask：00111100 进行位与操作，只保留 num 中与 mask 的 1 部分相对应的位。

01011011 &
00111100
--------
00011000

利用 x | 0s = x 和 x | 1s = 1s 的特点，可以实现设值操作。一个数 num 与 mask：00111100 进行位或操作，将 num 中与 mask 的 1 部分相对应的位都设置为 1。

01011011 |
00111100
--------
01111111

**位与运算技巧**

n&(n-1) 去除 n 的位级表示中最低的那一位 1。例如对于二进制表示 01011011，减去 1 得到 01011010，这两个数相与得到 01011010。

01011011 &
01011010
--------
01011010

n&(-n) 得到 n 的位级表示中最低的那一位 1。-n 得到 n 的反码加 1，也就是 -n=~n+1。例如对于二进制表示 10110100，-n 得到 01001100，相与得到 00000100。

10110100 &
01001100
--------
00000100

n-(n&(-n)) 则可以去除 n 的位级表示中最低的那一位 1，和 n&(n-1) 效果一样。

**移位运算**

\>\> n 为算术右移，相当于除以 2n，例如 -7 \>\> 2 = -2。

11111111111111111111111111111001 >> 2
--------
11111111111111111111111111111110

\>\>\> n 为无符号右移，左边会补上 0。例如 -7 \>\>\> 2 = 1073741822。

11111111111111111111111111111001 >>> 2
--------
00111111111111111111111111111111

<< n 为算术左移，相当于乘以 2n。-7 << 2 = -28。

11111111111111111111111111111001 << 2
--------
11111111111111111111111111100100

**mask 计算**

要获取 111111111，将 0 取反即可，~0。

要得到只有第 i 位为 1 的 mask，将 1 向左移动 i-1 位即可，1<<(i-1) 。例如 1<<4 得到只有第 5 位为 1 的 mask ：00010000。

要得到 1 到 i 位为 1 的 mask，(1<<i)-1 即可，例如将 (1<<4)-1 = 00010000-1 = 00001111。

要得到 1 到 i 位为 0 的 mask，只需将 1 到 i 位为 1 的 mask 取反，即 ~((1<<i)-1)。

**Java 中的位操作**

static int Integer.bitCount(); // 统计 1 的数量
static int Integer.highestOneBit(); // 获得最高位
static String toBinaryString(int i); // 转换为二进制表示的字符串

## 1. 统计两个数的二进制表示有多少位不同

1. Hamming Distance (Easy)

[Leetcode](https://leetcode.com/problems/hamming-distance/) / [力扣](https://leetcode-cn.com/problems/hamming-distance/)

Input: x = 1, y = 4

Output: 2

Explanation:
1 (0 0 0 1)
4 (0 1 0 0)
 ↑ ↑

The above arrows point to positions where the corresponding bits are different.

对两个数进行异或操作，位级表示不同的那一位为 1，统计有多少个 1 即可。

public int hammingDistance(int x, int y) {
 int z = x ^ y;
 int cnt = 0;
 while(z != 0) {
 if ((z & 1) == 1) cnt++;
 z = z >> 1;
 }
 return cnt;
}

使用 z&(z-1) 去除 z 位级表示最低的那一位。

public int hammingDistance(int x, int y) {
 int z = x ^ y;
 int cnt = 0;
 while (z != 0) {
 z &= (z - 1);
 cnt++;
 }
 return cnt;
}

可以使用 Integer.bitcount() 来统计 1 个的个数。

public int hammingDistance(int x, int y) {
 return Integer.bitCount(x ^ y);
}

## 2. 数组中唯一一个不重复的元素

136. Single Number (Easy)

[Leetcode](https://leetcode.com/problems/single-number/description/) / [力扣](https://leetcode-cn.com/problems/single-number/description/)

Input: [4,1,2,1,2]
Output: 4

两个相同的数异或的结果为 0，对所有数进行异或操作，最后的结果就是单独出现的那个数。

public int singleNumber(int[] nums) {
 int ret = 0;
 for (int n : nums) ret = ret ^ n;
 return ret;
}

## 3. 找出数组中缺失的那个数

268. Missing Number (Easy)

[Leetcode](https://leetcode.com/problems/missing-number/description/) / [力扣](https://leetcode-cn.com/problems/missing-number/description/)

Input: [3,0,1]
Output: 2

题目描述：数组元素在 0-n 之间，但是有一个数是缺失的，要求找到这个缺失的数。

public int missingNumber(int[] nums) {
 int ret = 0;
 for (int i = 0; i < nums.length; i++) {
 ret = ret ^ i ^ nums[i];
 }
 return ret ^ nums.length;
}

## 4. 数组中不重复的两个元素

260. Single Number III (Medium)

[Leetcode](https://leetcode.com/problems/single-number-iii/description/) / [力扣](https://leetcode-cn.com/problems/single-number-iii/description/)

两个不相等的元素在位级表示上必定会有一位存在不同。

将数组的所有元素异或得到的结果为不存在重复的两个元素异或的结果。

diff &= -diff 得到出 diff 最右侧不为 0 的位，也就是不存在重复的两个元素在位级表示上最右侧不同的那一位，利用这一位就可以将两个元素区分开来。

public int[] singleNumber(int[] nums) {
 int diff = 0;
 for (int num : nums) diff ^= num;
 diff &= -diff; // 得到最右一位
 int[] ret = new int[2];
 for (int num : nums) {
 if ((num & diff) == 0) ret[0] ^= num;
 else ret[1] ^= num;
 }
 return ret;
}

## 5. 翻转一个数的比特位

190. Reverse Bits (Easy)

[Leetcode](https://leetcode.com/problems/reverse-bits/description/) / [力扣](https://leetcode-cn.com/problems/reverse-bits/description/)

public int reverseBits(int n) {
 int ret = 0;
 for (int i = 0; i < 32; i++) {
 ret <<= 1;
 ret |= (n & 1);
 n >>>= 1;
 }
 return ret;
}

如果该函数需要被调用很多次，可以将 int 拆成 4 个 byte，然后缓存 byte 对应的比特位翻转，最后再拼接起来。

private static Map<Byte, Integer> cache = new HashMap<>();

public int reverseBits(int n) {
 int ret = 0;
 for (int i = 0; i < 4; i++) {
 ret <<= 8;
 ret |= reverseByte((byte) (n & 0b11111111));
 n >>= 8;
 }
 return ret;
}

private int reverseByte(byte b) {
 if (cache.containsKey(b)) return cache.get(b);
 int ret = 0;
 byte t = b;
 for (int i = 0; i < 8; i++) {
 ret <<= 1;
 ret |= t & 1;
 t >>= 1;
 }
 cache.put(b, ret);
 return ret;
}

## 6. 不用额外变量交换两个整数

程序员代码面试指南 ：P317

a = a ^ b;
b = a ^ b;
a = a ^ b;

## 7. 判断一个数是不是 2 的 n 次方

231. Power of Two (Easy)

[Leetcode](https://leetcode.com/problems/power-of-two/description/) / [力扣](https://leetcode-cn.com/problems/power-of-two/description/)

二进制表示只有一个 1 存在。

public boolean isPowerOfTwo(int n) {
 return n > 0 && Integer.bitCount(n) == 1;
}

利用 1000 & 0111 == 0 这种性质，得到以下解法：

public boolean isPowerOfTwo(int n) {
 return n > 0 && (n & (n - 1)) == 0;
}

## 8. 判断一个数是不是 4 的 n 次方

342. Power of Four (Easy)

[Leetcode](https://leetcode.com/problems/power-of-four/) / [力扣](https://leetcode-cn.com/problems/power-of-four/)

这种数在二进制表示中有且只有一个奇数位为 1，例如 16（10000）。

public boolean isPowerOfFour(int num) {
 return num > 0 && (num & (num - 1)) == 0 && (num & 0b01010101010101010101010101010101) != 0;
}

也可以使用正则表达式进行匹配。

public boolean isPowerOfFour(int num) {
 return Integer.toString(num, 4).matches("10\*");
}

## 9. 判断一个数的位级表示是否不会出现连续的 0 和 1

693. Binary Number with Alternating Bits (Easy)

[Leetcode](https://leetcode.com/problems/binary-number-with-alternating-bits/description/) / [力扣](https://leetcode-cn.com/problems/binary-number-with-alternating-bits/description/)

Input: 10
Output: True
Explanation:
The binary representation of 10 is: 1010.

Input: 11
Output: False
Explanation:
The binary representation of 11 is: 1011.

对于 1010 这种位级表示的数，把它向右移动 1 位得到 101，这两个数每个位都不同，因此异或得到的结果为 1111。

public boolean hasAlternatingBits(int n) {
 int a = (n ^ (n >> 1));
 return (a & (a + 1)) == 0;
}

## 10. 求一个数的补码

476. Number Complement (Easy)

[Leetcode](https://leetcode.com/problems/number-complement/description/) / [力扣](https://leetcode-cn.com/problems/number-complement/description/)

Input: 5
Output: 2
Explanation: The binary representation of 5 is 101 (no leading zero bits), and its complement is 010. So you need to output 2.

题目描述：不考虑二进制表示中的首 0 部分。

对于 00000101，要求补码可以将它与 00000111 进行异或操作。那么问题就转换为求掩码 00000111。

public int findComplement(int num) {
 if (num == 0) return 1;
 int mask = 1 << 30;
 while ((num & mask) == 0) mask >>= 1;
 mask = (mask << 1) - 1;
 return num ^ mask;
}

可以利用 Java 的 Integer.highestOneBit() 方法来获得含有首 1 的数。

public int findComplement(int num) {
 if (num == 0) return 1;
 int mask = Integer.highestOneBit(num);
 mask = (mask << 1) - 1;
 return num ^ mask;
}

对于 10000000 这样的数要扩展成 11111111，可以利用以下方法：

mask |= mask >> 1 11000000
mask |= mask >> 2 11110000
mask |= mask >> 4 11111111

public int findComplement(int num) {
 int mask = num;
 mask |= mask >> 1;
 mask |= mask >> 2;
 mask |= mask >> 4;
 mask |= mask >> 8;
 mask |= mask >> 16;
 return (mask ^ num);
}

## 11. 实现整数的加法

371. Sum of Two Integers (Easy)

[Leetcode](https://leetcode.com/problems/sum-of-two-integers/description/) / [力扣](https://leetcode-cn.com/problems/sum-of-two-integers/description/)

a ^ b 表示没有考虑进位的情况下两数的和，(a & b) << 1 就是进位。

递归会终止的原因是 (a & b) << 1 最右边会多一个 0，那么继续递归，进位最右边的 0 会慢慢增多，最后进位会变为 0，递归终止。

public int getSum(int a, int b) {
 return b == 0 ? a : getSum((a ^ b), (a & b) << 1);
}

## 12. 字符串数组最大乘积

318. Maximum Product of Word Lengths (Medium)

[Leetcode](https://leetcode.com/problems/maximum-product-of-word-lengths/description/) / [力扣](https://leetcode-cn.com/problems/maximum-product-of-word-lengths/description/)

Given ["abcw", "baz", "foo", "bar", "xtfn", "abcdef"]
Return 16
The two words can be "abcw", "xtfn".

题目描述：字符串数组的字符串只含有小写字符。求解字符串数组中两个字符串长度的最大乘积，要求这两个字符串不能含有相同字符。

本题主要问题是判断两个字符串是否含相同字符，由于字符串只含有小写字符，总共 26 位，因此可以用一个 32 位的整数来存储每个字符是否出现过。

public int maxProduct(String[] words) {
 int n = words.length;
 int[] val = new int[n];
 for (int i = 0; i < n; i++) {
 for (char c : words[i].toCharArray()) {
 val[i] |= 1 << (c - 'a');
 }
 }
 int ret = 0;
 for (int i = 0; i < n; i++) {
 for (int j = i + 1; j < n; j++) {
 if ((val[i] & val[j]) == 0) {
 ret = Math.max(ret, words[i].length() \* words[j].length());
 }
 }
 }
 return ret;
}

## 13. 统计从 0 ~ n 每个数的二进制表示中 1 的个数

338. Counting Bits (Medium)

[Leetcode](https://leetcode.com/problems/counting-bits/description/) / [力扣](https://leetcode-cn.com/problems/counting-bits/description/)

对于数字 6(110)，它可以看成是 4(100) 再加一个 2(10)，因此 dp[i] = dp[i&(i-1)] + 1;

public int[] countBits(int num) {
 int[] ret = new int[num + 1];
 for(int i = 1; i <= num; i++){
 ret[i] = ret[i&(i-1)] + 1;
 }
 return ret;
}