Bit Manipulation, 进制转换

#G面试准备

CC150 - Bit Manipulation

Common Bit Tasks: Getting and Setting

The following operations are very important to know, but do not simply memorize them. Memorizing leads to mistakes that are impossible to recover from. Rather, understand *how* to implement these methods, so that you can implement these, and other, bit problems.

Get Bit

This method shifts 1 over by `i` bits, creating a value that looks like `00010000`. By performing an AND with num, we clear all bits other than the bit at bit `i`. Finally, we compare that to `0`. If that new value is not zero, then bit `i` must have a 1. Otherwise, bit `i` is a 0.

```
1  boolean getBit(int num, int i) {
2    return ((num & (1 << i)) != 0);
3  }
```

Set Bit

`SetBit` shifts 1 over by `i` bits, creating a value like `00010000`. By performing an OR with num, only the value at bit `i` will change. All other bits of the mask are zero and will not affect num.

```
1  int setBit(int num, int i) {
2    return num | (1 << i);
3  }
```

Clear Bit

This method operates in almost the reverse of `setBit`. First, we create a number like `11101111` by creating the reverse of it (`00010000`) and negating it. Then, we perform an AND with num. This will clear the `i`th bit and leave the remainder unchanged.

```
1  int clearBit(int num, int i) {
2    int mask = ~(1 << i);
3    return num & mask;
4  }
```

To clear all bits from the most significant bit through `i` (inclusive), we create a mask with a `1` at the `i`th bit (`1 << i`). Then, we subtract `1` from it, giving us a sequence of `0`s followed by `i` `1`s. We then AND our number with this mask to leave just the last `i` bits.

```
1  int clearBitsMSBthroughI(int num, int i) {
2    int mask = (1 << i) - 1;
3    return num & mask;
4  }
```

To clear all bits from `i` through `0` (inclusive), we take a sequence of all `1`s (which is `-1`) and shift it left by `i + 1` bits. This gives us a sequence of 1s (in the most significant bits) followed by `i` `0` bits.

```
1  int clearBitsIthrough0(int num, int i) {
2    int mask = (-1 << (i + 1));
3    return num & mask;
4  }
```

Update Bit

To set the `i`th bit to a value `v`, we first clear the bit at position `i` by using a mask that looks like `11101111`. Then, we shift the intended value, `v`, left by `i` bits. This will create a number with bit `i` equal to `v` and all other bits equal to `0`. Finally, we OR these two numbers, updating the `i`th bit if `v` is `1` and leaving it as `0` otherwise.

```
1   int updateBit(int num, int i, boolean bitIs1) {
2     int value = bitIs1 ? 1 : 0;
3     int mask = ~(1 << i);
4     return (num & mask) | (value << i);
5   }
```

面经题

> 第一题是给一个int，返回它的二进制的最高位1的Index。比如，对于int 4,二进制是100，最高位1的index就是2；对于int 1，二进制是1，最高位的index是0；对于0，返回-1；对于负数，因为最高位都是符号位，所以返回31。解法就是对于正数，不断的除2，然后计算除多少次能得到0，对于负数返回1，对于0返回-1。

- 用到以上的getBit

```
// input: num
for (int i = 31; i >= 0; i--) {
  int mask = 1 << i;
  if (num & mask != 0) return i;
}
return -1;
```

421. Maximum XOR of Two Numbers in an Array

Maximum XOR of Two Numbers in an Array - LeetCode

- 几个关键性质:
  - A ^ B == C, C ^ B == A, C ^ A == B
  - 让最大位到第i位都为1 初始化mask为0 i从31开始递减:

`mask = mask | (1 << i); // set 1 from msb to i`

  - 设置第i位为1:

`int tmp = max | (1 << i); // set ith bit to 1`

- 31次循环 每次都用set来存候选的不同前缀(从MSB开始往前)
- 先暂时设置tmp当前第i位为1 然后去候选set里遍历num 看是否存在一个num2 使得num ^ num2 == tmp
  - 找的方法就是`set.contains(tmp ^ num)`
  - 如果存在 就把当前的max改为tmp 当前第i位就是1了

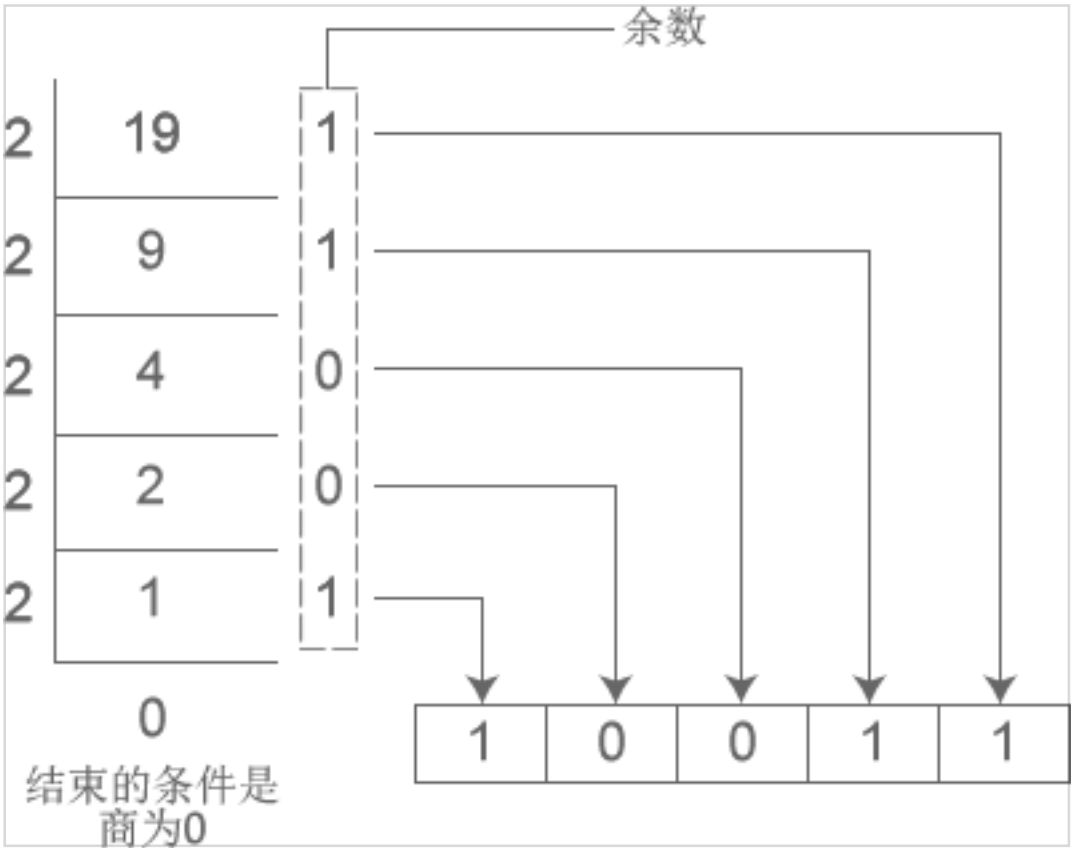
    - 否则max值不变 当前第i位还是0
- 一直循环到最后就可以把最大的两个数异或值找到 输出max

进制转换

- 三进制转十进制
    - 122 ➡ 3^2 * 1 + 3^1 * 2 + 3^0 * 2
    - 简化计算: ((1 * 3) + 2) * 3) + 2
- 任意进制之间转换: a到b进制 先把a进制转换为10进制 再把10进制转换为b进制

> 十进制数temp 转为 N进制数
> 将temp 不断的除以N,求模，直到temp = 0,然后将每次求的模 倒序输出即可。

- 十进制转换为二进制例子: 辗除法



> 如图所示，以2为除数，一直相除下去，直到商为0，余数则为求得的二进制数
> 注意：余数要倒序排列，也就是说，最先求得的余数排在二进制的最后面，最后求得的余数排在二进制的最前面。上面的例子中，最后求得的二进制数为 10011。