# 背景

不安全网页的黑名单包含100亿个黑名单网页，每个网页的URL最多占用64字节。现在想要实现一种网页过滤系统，可以根据网页的URL判断该网页是否在黑名单上，请设计该系统，要求该系统允许有万分之一以下的判断错误率，并且使用的额外空间不要超过30G。



对于涉及网页黑名单系统，垃圾邮件过滤系统，爬虫的网址判断重复系统，同时可以容忍一定程度的失误率，对空间要求较严格的情况，可以使用布隆过滤器。

# 位运算符

位运算符：&、|、^、~ 、<<、>>、>>>

# 布隆过滤器

## 概述

布隆过滤器可以精确地代表一个集合（不是准确代表集合，精确程度由用户的具体设计决定，做到100%的正确是不可能的），可精确（不是准确）判断某一元素是否在此集合中。

布隆过滤器的优势在于，利用很少的空间可以做到精确率较高。

## 原理







注：如果a对应的bitarray中所有位置是否为1，如果有1个不为1则一定不在集合里，如果都为1，则a在该集合中，但是可能误判。

## 参数

布隆过滤器的bitarray大小如何确定？

说明：大小为m，样本数量为n，失误率为p。

n=100亿，p=0.01%：

单个样本大小不影响布隆过滤大小，只影响了哈希函数的实现细节：

求得m=19.19n，向上取整为m=20n。

2000亿bit，约为25G。

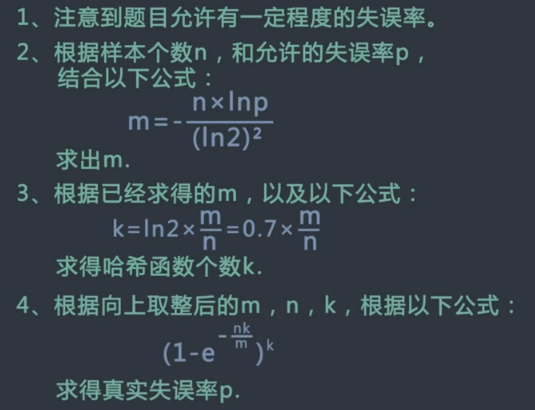
求得：k=14

综上，m=20n，k=14

求得真实失误率p=0.006%，占用大小为25G。

总结：

1. 注意到题目允许一定程度的失误率；
2. 根据样本个数n，和允许的失误率p，结合以下公式求出m：
3. 根据已经求得的m，结合以下公式求得哈希函数个数k：
4. 根据向上取整后的m，n，k，结合以下公式求得真实失误率p：



# 应用

## 不用加号的加法

**题目：**

设计一个函数把两个数字相加。不得使用 + 或者其他算术运算符。

示例:

输入: a = 1, b = 1

输出: 2

提示：

a, b 均可能是负数或 0

结果不会溢出 32 位整数

**分析：**

**代码：**

## 两整数之和

## 交换整数值

**题目：**如何不用任何额外变量交换两个整数的值？

分析：

对于给定整数a和b，如果采用额外申请内存的方式：

int c = a;

a = b;

b = c;

假设a=a0，b=b0：

a = a ^ b；🡪a = a0 ^ b0，b = b0

b = a ^ b；🡪a = a0 ^ b0，b = a0 ^ b0 ^ b0 = a0

a = a ^ b；🡪a = a0 ^ b0 ^ a0 = b0，b = a0

代码：

class Solution {

public:

vector<int> swapNumbers(vector<int>& numbers) {

numbers[0] = numbers[0] ^ numbers[1];

numbers[1] = numbers[0] ^ numbers[1];

numbers[0] = numbers[0] ^ numbers[1];

return numbers;

}

};

## 插入

## 求较大值

**题目：**给定两个32位整数a和b，返回a和b中较大的，但是不能用任何比较判断。

**分析：**

**方法一：得到a-b的符号，根据该符号决定返回a或b**

public static int flip(int n){

return n ^ 1;

}

public static int sign(int n){

return flip((n>>31)&1);

}

public static int getMax(int a,int b){

int c = a - b;

int scA = sign(c);

int scB = flip(scA);

return a\*scA + b\*scB;

}

注：方法一可能存在问题，当a-b溢出时，会发生错误。

**方法二：**

public static int getMax2(int a,int b){

int a = a - b;

int as = sign(a); //a的符号,as==1表示a为非负,as==0表示a为负

int bs = sign(b); //b的符号,bs==1表示b为非负,bs==0表示b为负

int cs = sign(c); //a-b的符号

int difab = as ^ bs; //表示a和b是否符号相同,相同为0,不相同为1

int sameab = flip(difab); //表示aheb是否符号相同,相同为1,不同为0

int returnA = difab\*as + sameab\*cs;

int returnB = flip(returnA);

return a\*returnA + b\*returnB;

}

## 消失数字

## 缺失数字

## 只出现一次的数字

## 数组中出现次数超一半的数字

**题目：**

数组中有一个数字出现的次数超过数组长度的一半，请找出这个数字。

你可以假设数组是非空的，并且给定的数组总是存在多数元素。

示例 1:

输入: [1, 2, 3, 2, 2, 2, 5, 4, 2]

输出: 2

限制：

1 <= 数组长度 <= 50000

**分析：**

1. 排序

排序后中间的元素一定是出现超过一半的数字。

1. 位运算

给出输入[1,2,3,3,3]，将这几个数字写成二进制数(整型32位，这里只写出后面4位)[0001, 0010, 0011, 0011, 0011]。在这4位二进制数字中，各自的最高位分别为[0，0，0，0]，则最高位出现最多的数字是二进制0。同理，高二位出现最多的二进制数字为0，高三位出现最多的二进制数字为1，最低位出现最多的二进制数字为1。则4位出现最多的二进制数字拼起来为0011，值为3，即结果。

注：位运算的优点在于占用内存少且运算速度快！

1. 哈希表

使用unordered\_map数据结构，扫一遍给定数组，存储每个数字出现的次数。

再扫一遍哈希表，找出出现次数超过一半的数字

**代码：**

class Solution {

public:

int majorityElement(vector<int>& nums) {

int ans = 0;

int n = nums.size();

//位运算

for(int i=0; i < 32; i++){

int cnt = 0;

for(int n: nums){

if(n & 1 << i) cnt ++;

}

if(cnt > n / 2) ans ^= 1 << i;

}

return ans;

}

};

class Solution {

public:

int majorityElement(vector<int>& nums) {

//方法1：排序后中间的元素一定是出现超过一半的数字

sort(nums.begin(),nums.end());

return nums[nums.size()/2];

//方法2：哈希表

unordered\_map<int,int>mp;

for(auto it : nums){

mp[it]++;

if(mp[it]>nums.size()/2) return it;

}

return 0;

//方法3：超过一半的数字比其他所有数字的总和次数多

int n=1;

int result=nums[0];

for(int i=1;i<nums.size();i++){

if(n==0){

result=nums[i];

n=1;

}

else if(result==nums[i])n++;

else n--;

}

return result;

}

};

## 查找奇数次数字

**题目：**给定一个整型数组arr，其中只有一个数出现了奇数次，其他的数都出现了偶数次，请打印这个数。要求时间复杂度为O(N)，额外空间复杂度为O(1)。

**分析：**

N与0异或结果为n，n与n异或结果为0

E0=0,arr=[C,B,D,A,A,B,C]

异或运算满足交换律和结合律

按照原始arr数出现的顺序异或结果，与该数组异或的结果相同：[A,A,B,B,C,C,D]

e0与上述数组异或的结果为e0=D

**拓展：**给定一个整型数组arr，其中有两个数出现了奇数次，其他的数都出现了偶数次，请打印这两个数。要求时间复杂度为O(N)，额外空间复杂度为O(1)。



注：对于数组中存在奇偶个数或者奇偶相关的查找排序操作，可以采用位运算解决。

## 找不同

## 加密解密

异或运算可完成简单的加密与解密过程：

明文text，用户给定的密码pw，假设密文为cipher：

cipher = text ^ pw

text = cipher ^ pw = (text ^ pw) ^ pw

= text ^ (pw ^ pw) = text

如果text长度大于pw，循环使用pw与text进行按位异或。