# 描述

给定一个非空整数数组，除了某个元素只出现一次以外，其余每个元素均出现两次。找出那个只出现了一次的元素。

**说明：**

你的算法应该具有线性时间复杂度。 你可以不使用额外空间来实现吗？

**示例 1:**

输入: [2,2,1]

输出: 1

**示例 2:**

输入: [4,1,2,1,2]

输出: 4

# 分析

比较通用的方法：快慢指针+哈希表

## 方法一：暴力求解

**思路：**列表操作

遍历nums中的每一个元素，如果某个nums中的数字是新出现的，则将它添加到列表中，如果某个数字已经在列表中，删除它。

**代码：**

//哈希暴力求解

class Solution {

public:

int singleNumber(vector<int>& nums) {

map<int,int> m;

for(int i=0; i<nums.size();i++){

if(m.find(nums[i]) != m.end()){

m.erase(m.find(nums[i]));

}else{

m[nums[i]] = 1;

}

}

return m.begin()->first;

}

};

## 方法二：快慢指针

**思路：**

先排序，再用双指针对比。

**代码：**

class Solution {

public:

int singleNumber(vector<int>& nums) {

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

for(int i = 0, j = 1; j < nums.size(); i += 2, j += 2){

if(nums[i] != nums[j]) return nums[i];

}

return nums[nums.size() - 1];

}

};

## 方法三：哈希表

**思路：**

我们用哈希表避免每次查找元素是否存在需要的O(n)时间。

1、遍历nums中的每一个元素

2、查找hash\_table中是否有当前元素的键

3、如果没有，将当前元素作为键插入hash\_table

4、最后，hash\_table中仅有一个元素，用popitem获得它

**代码：**

class Solution {

public:

int singleNumber(vector<int>& nums) {

unordered\_map<int,int> mp;

for(int i=0;i<nums.size();i++)

{

mp[nums.at(i)]++;

}

unordered\_map<int,int>::iterator it = mp.begin();

for(; it!=mp.end(); it++)

{

if(it->second==1)

return it->first;

}

return 0;

}

};

## 方法四：异或法

**思路：**两个相等的数字经过异或后，还原为0，所以最后留下的就是只有一个的数字了。

如果我们对 0 和二进制位做 XOR 运算，得到的仍然是这个二进制位

a⊕0 =a

如果我们对相同的二进制位做 XOR 运算，返回的结果是 0

a⊕a=0

XOR 满足交换律和结合律

a⊕b⊕a=(a⊕a)⊕b=0⊕b=b

所以我们只需要将所有的数进行 XOR 操作，得到那个唯一的数字。

**代码：**

//异或

class Solution {

public:

int singleNumber(vector<int>& nums) {

int v=0;

for(int i=0; i<nums.size(); i++){

v ^= nums[i];

}

return v;

}

};