在该笔记中，仅仅对我认为有意思以及错误多次的例题进行讲解，分Tags进行描述

# Array

注意其中的vector::size() 返回的为size\_t为无符号整数。乘-1\*num比-num慢很多

* 1. 在不适用额外空间的条件下进行数据处理。

448. Find All Numbers Disappeared in an Array – 采用变形的桶排序进行处理。

1.2 统计数组中元素出现次数大于一半的数据（不同于统计出现次数最多的元素）

该文章发表于1991年，MJRTY - A Fast Majority Vote Algorithm。我的思路是排序或者采用map进行处理。但通过分析数据可发现，如果数据中存在元素等于majority的个数大于向下取整[N/2]，那么可采用动态规划的方式进行O(N)处理，其基于majority连续的个数大于其它元素连续的个数。

* 1. 寻找数组中的重复的数字(217 Contains Duplicate )

解决方法直接的是采用hashmap的结构进行处理，或者采用排序遍历的方式，这两种可以记录所有的重复数字，前者的时间效率更高。

如果只是随意输出任一重复的数字，可采用快速排序+过程中比较，或自建in-house的set，有人采用如下code1.2解决方法，还未看懂

(Given an array of integers, find if the array contains any duplicates. Your function should return true if any value appears at least twice in the array, and it should return false if every element is distinct.)

* 1. 比较数组中重复的数字(697. Degree of an Array )

因为题中说元素值为0-50000，顾采用数组作为哈希map，但是我错误的采用了三个map进行记录—元素频率、开始值、结束值。但是其实可以采用两个map解决问题，元素频率和起始位置。

* 1. 去除冗余计算（661 Image Smoother），需要简化代码以及提高程序的扩展性，注意函数中两个形参引用在相同时的特殊情况。
  2. 数组中元素值区间为[0, N]，数组长度为N，且各个元素互不相等，

268. Missing Number。只能增加固定的额外存储空间。

我最开始的想法是通过桶排序的变形（将对应元素设置为负值），但是看了discuss后看到两种有趣的解法，第一种，总的值( (N)/(N+1)/2)减去所有元素的和，其中注意计算总值之前将数据转换为long long int；第二种，采用按位异或的方式，按位异或有四个性质：0 xor a = a xor 0 == a; a xor b = b xor a; a xor b xor c = (a xor b) xor c = a xor ( b xor c ); d = a xor b xor c < -- > a = d xor b .xor c。那么缺失数据就可以通过异或进行解决了，假设则缺少的元素值为N，异或为，采用该原理得出缺失的元素值。

* 1. 元素按序间的最大差值，121. Best Time to Buy and Sell Stock，将该问题抽象为最大逆序间隔，fmin(i)表示数组[0-i]中的最小元素，fmax(i)表示最小元素后的最大元素，在每次改变最大元素的值时，通过比较对最大间隔进行更新。
  2. 合并两个排序数组，88. Merge Sorted Array，通过前向将两个数组合并比较困难，这时候需要从反向进行处理。

附录code：

1.2

struct Node

{

int val;

struct Node \*next;

};

struct Set

{

int bucketSize;

struct Node \*\*table;

};

void initSet(struct Set \*set, int bucketSize)

{

set->bucketSize = bucketSize;

set->table = malloc(sizeof(struct Node\*) \* bucketSize);

memset(set->table, 0, sizeof(struct Node\*) \* bucketSize);

}

bool addValue(struct Set \*s, int val)

{

int idx = val > 0 ? val : -val;

idx %= s->bucketSize;

struct Node \*ptr = s->table[idx];

while(ptr != NULL)

{

if(ptr->val == val)

{

return false;

}

ptr = ptr->next;

}

ptr = malloc(sizeof(struct Node));

ptr->val = val;

ptr->next = s->table[idx];

s->table[idx] = ptr;

return true;

}

void releaseSet(struct Set \*s)

{

struct Node \*ptr, \*tmp;

for(int i = 0; i < s->bucketSize; ++i)

{

ptr = s->table[i];

while(ptr != NULL)

{

tmp = ptr;

ptr = ptr->next;

free(tmp);

}

}

free(s->table);

s->table = NULL;

s->bucketSize = 0;

}

bool containsDuplicate(int\* nums, int numsSize) {

if(numsSize < 2)

{

return false;

}

struct Set set;

initSet(&set, numsSize / 2);

for(int i = 0; i < numsSize; ++i)

{

if(!addValue(&set, nums[i]))

{

releaseSet(&set);

return true;

}

}

releaseSet(&set);

return false;

}

# vector增删查改

增 –

在该iteratot之前插入元素，返回第一个插入的元素的itreator

iterator insert (const\_iterator position, const value\_type& val);

删 –

删除该元素，返回删除最后的元素的itreator

iterator erase( iterator pos );

map：

iterator map::find(val)