/\*

Given an integer array of size n, find all elements that appear more than ⌊ n/3 ⌋ times. The algorithm should run in linear time and in O(1) space.

way-1:用了额外的空间

way-2:

借助于Majority Element中的Boyer Moore算法，可以在O(1)的空间和线性时间内求解问题。

因为出现次数大于n/3的元素最多只有两个，所以最开始可以维护两个数字(num1,num2)和两个计数器(counter1,counter2)。

遍历数组，

当数组中元素和num1或者num2相同，对应的counter1或者counter2加1；

如果counter1或counter2为0，将遍历到的该元素赋给num1或者nums2；

否则counter1和counter2都减1。

其中第3步是算法的核心，就是每次删除三个不相同的数，最后留下的一定是出现次数超过1/3的数，这个思想可以推广到出现次数超过1/k次的元素有哪些。

这个方法也叫三三抵消，最后会剩下两个candidates，但是注意此时不是谁占多数谁是最终结果，反例[1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 4, 4] 三三抵消后剩下 [1, 4，4] 4数量占优，但结果应该是1，所以三三抵消后，再loop一遍找1和4谁数量超过了len(nums)/3

\*/

class Solution {

public:

vector<int> majorityElement(vector<int>& nums)

{

//way-1

/\*

vector<int> ret;

map<int,int> mm;

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

{

mm[nums[i]]++;

}

map<int,int>::iterator it;

for(it=mm.begin();it!=mm.end();it++)

{

if(it->second>nums.size()/3)

ret.push\_back(it->first);

}

return ret;

\*/

//way-2

vector<int> ret;

int num1,num2;

int count1=0,count2=0;

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

{

if(num1==nums[i])

count1++;

else if(num2==nums[i])

count2++;

else if(count1==0)

{

num1=nums[i];

count1++;

}

else if(count2==0)

{

num2=nums[i];

count2++;

}

else

{

count1--;

count2--;

}

}

count1=0;

count2=0;

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

{

if(nums[i]==num1)

count1++;

else if(nums[i]==num2)

count2++;

}

if(count1>nums.size()/3)

ret.push\_back(num1);

if(count2>nums.size()/3)

ret.push\_back(num2);

return ret;

}

};