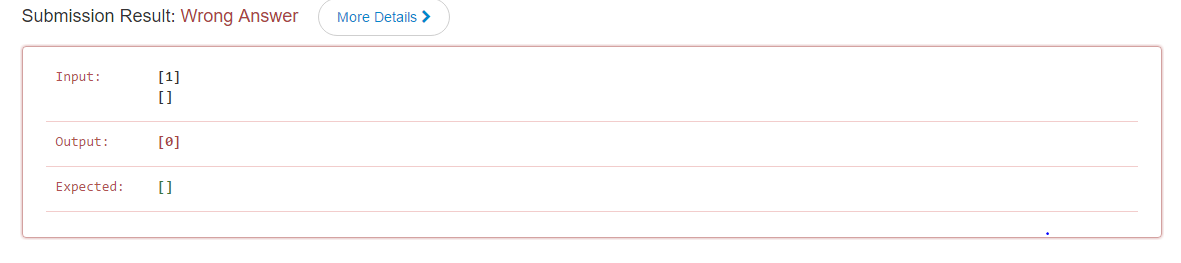
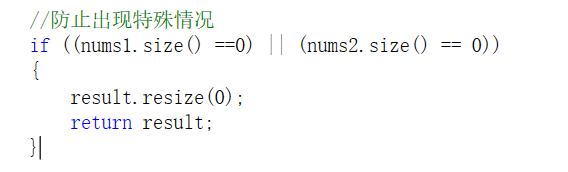
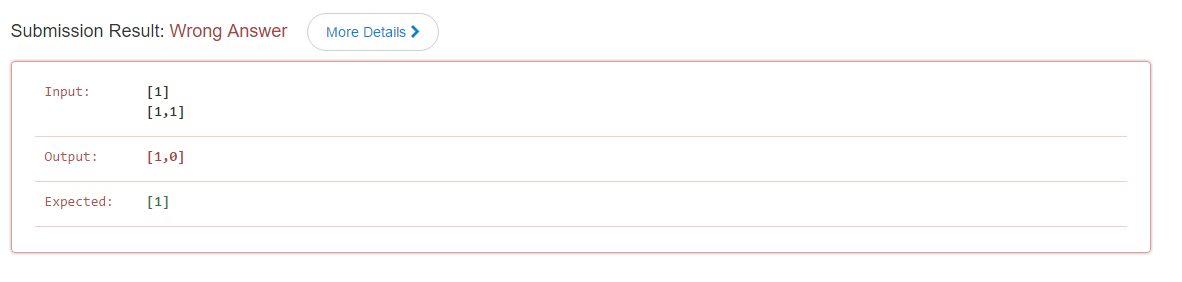
## 第一次提交错误：



由于返回值result参数是按照nums1的长度初始化的，因此在该测试数据下返回的是初始化的值，而不是空数组。解决方法是讲此特殊情况下的返回值清空后才返回，代码如下：

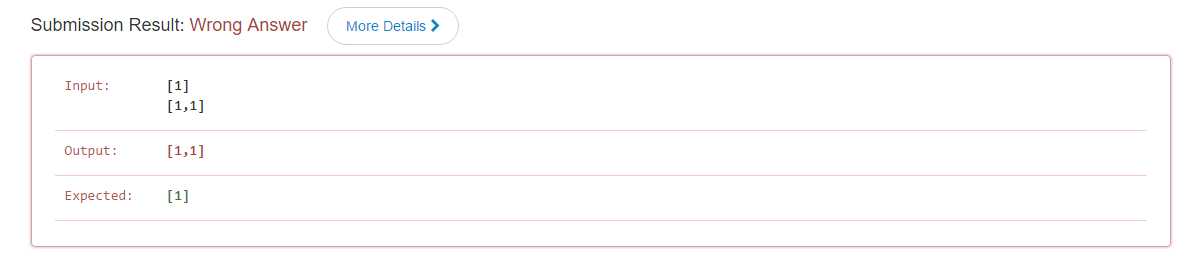


## 第二次提交错误：



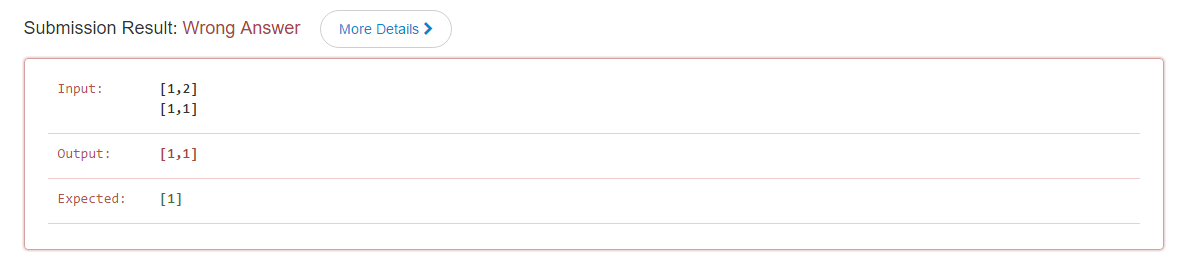
由于result的长度是按照nums1的长度初始化的，所以当nums1.size()<nums2.size()时会出现运行时错误。解决方法是讲result按照两个形参中较长的数组长度作为返回数组的长度即可。

## 第三次提交错误：



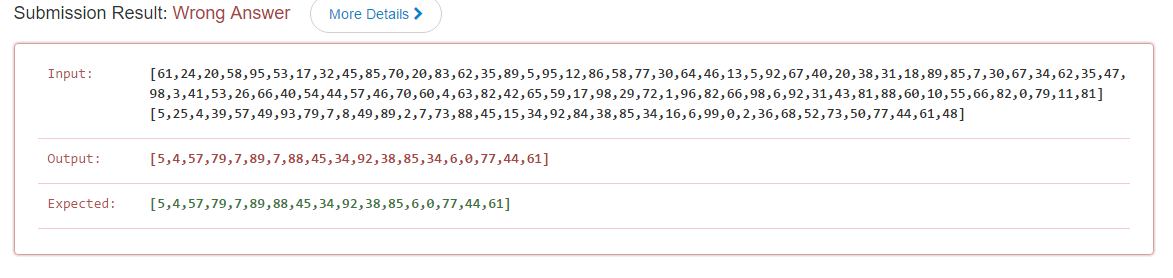
原始思路有问题。原始思路是想将其中一个数组中的重复元素剔除后再进行比较以免出现重复比较的情况，但是对于剔除哪个数组的重复元素则没有区分，导致无法处理测试样例。

第四次提交错误：



思路有缺陷，未能考虑完整。仅以数组长短来确定比较对象想法太过简单。该问题与第三次提交错误的本质原因相似。目前思路是讲长短数组相互进行两次比较，取较短的结果数组作为最终的值。

第五次提交错误：



上述方法并不能阻止错误结果泄漏到最终结果中。新的解决思路是首先对两个待比较的数组的元素做统计，如下表所示：

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| a[] | 2 | 3 | 1 | 1 | 2 | 3 | 1 | 2 | 2 |
| 统计值 | 4 | 2 | 3 | 3 | 4 | 2 | 3 | 4 | 4 |

然后剔除其中一个数组及其统计结果中的重复值最后再进行比较。该思路执行有效，但性能较差，代码如下

class Solution {

public:

vector<int> intersect(vector<int>& nums1, vector<int>& nums2)

{

vector<int> sta\_nums1(nums1),sta\_nums2(nums2);

vector<int> result( (nums1.size() >= nums2.size())? nums1.size():nums2.size() );

int index\_nums1=0;

int index\_nums2=0;

int result\_len=0;

//统计数组nums1

int temp\_index,count;

while(index\_nums1 < nums1.size())

{

count=0;

for (temp\_index=0;temp\_index<nums1.size();temp\_index++)

{

if (nums1[index\_nums1] == nums1[temp\_index])

{

count ++;

}

}

sta\_nums1[index\_nums1] = count;

index\_nums1 ++;

}

//统计数组nums2并删除nums2和结果中的重复值

//如果N在之前的统计中已经出现过，则当前不再统计

int length=0;

vector<int> new\_nums2(nums2);

while(index\_nums2 < nums2.size())

{

count=0;

int index\_before=0;

while( (index\_before < index\_nums2) && (nums2[index\_before] != nums2[index\_nums2]) )

index\_before ++;

if (index\_before == index\_nums2)

{

for (temp\_index=0;temp\_index<nums2.size();temp\_index++)

{

if (nums2[index\_nums2] == nums2[temp\_index])

{

count ++;

}

}

new\_nums2[length] = nums2[index\_nums2];

sta\_nums2[length] = count;

length ++;

}

index\_nums2 ++;

}

sta\_nums2.resize(length);

new\_nums2.resize(length);

//开始比较

result\_len=0;

index\_nums1=0;

int index\_newn2=0;

int index\_result=0;

while(index\_newn2 < new\_nums2.size())

{

while( (index\_nums1 < nums1.size()) && (new\_nums2[index\_newn2] != nums1[index\_nums1]))

index\_nums1 ++;

if (index\_nums1 < nums1.size())

{

if (sta\_nums2[index\_newn2] <= sta\_nums1[index\_nums1])

{

result\_len += sta\_nums2[index\_newn2];

while(index\_result < result\_len)

{

result[index\_result] = new\_nums2[index\_newn2];

index\_result ++;

}

}

else

{

result\_len += sta\_nums1[index\_nums1];

while(index\_result < result\_len)

{

result[index\_result] = new\_nums2[index\_newn2];

index\_result ++;

}

}

}

index\_newn2 ++;

index\_nums1 = 0 ;

}

result.resize( result\_len );

return result;

}

};

性能结果

