查阅更多的题解，请点击

Problem

287. Find the Duplicate Number

Given an array nums containing n + 1 integers where each integer is between 1 and n (inclusive), prove that at least one duplicate number must exist. Assume that there is only one duplicate number, find the duplicate one.

Example 1:

```
Input: [1,3,4,2,2]
Output: 2
```

Example 2:

```
Input: [3,1,3,4,2]
Output: 3
```

Note:

- You must not modify the array (assume the array is read only).
- You must use only constant, O(1) extra space.
- Your runtime complexity should be less than O(n2).
- There is only one duplicate number in the array, but it could be repeated more than once.

Solution

solution one

O(n) time, O(n) space

看到该题的直观想法，对n+1个数的数组进行遍历，将每个数插入一个hash表，若插入时该元素已经在表中存在，则返回该元素，由题目要求，该元素一定存在。由于hash表的插入和查找都是O(1)，该算法最坏情况下需要遍历到数组尾，所以时间复杂度:O(n) 空间复杂度：O(n)。

solution two

O(nlogn) time, O(1) space

由于题目要求，空间开销要尽可能的小，所以需要不同于解法一的算法。由于输入数据结构为数组，考虑原问题是否可以reduce：一个很希望达成的reduce策略：将输入数据分成两组，一组中包含duplicate数据，一组不包含；这样，我们就将原问题reduce到了更小的问题，这样的策略很像广为人

知的二分查找。但不同的该问题输入是无序数据，如何对数据进行我们所希望的划分？注意题目中的条件：

- 设任意一个数为a_k,有$a_k \in [1,n]$
- n+1个数，仅存在一组重复数据

则可以根据(n+1)/2对输入数据进行划分，即duplicate one是在[1,(n+1)/2]还是在区间((n+1)/2,n]

solution three

O(n) time,o(1) space

该解法非常的优雅，是在LeetCode的讨论区看到的，该方法很巧妙，但理解起来并不容易.

对于题目给出的n+1个数组，若n+1个数互不相同，则数组下标和该数本身有一一对应关系，比如数组{2,1,3},有下标和数值的对应关系：0->2,1->1,2->3，该对应关系类似函数的一一映射，设数组下标为x，对应值为f(x),如果我们从下标为0出发，根据这个函数计算出一个值，以这个值为新的下标，再用这个函数计算，以此类推，直到下标超界。实际上可以产生一个类似链表一样的序列,该例中下标的序列为0->2-3；若如题目所言，存在这样的重复项，则无法实现一一映射，序列中也会出现环，找到该环，就可以找到重复项。

```
int slow = 0;
int fast = 0;
while (true)
{
    slow = nums[slow];
    fast = nums[nums[fast]];
    if (slow == fast)
        break;
}
```

上述代码中通过快慢指针，尝试寻找环(即出现$f(x_1) = x = f(x_2)$)，这里可能会引起疑惑，会不会我们找到属于$f(x) = x$的情况（姑且称之为自环）?不妨用反证法证明：

- 假设经过k步，出现f(x)=x的情况，上述查找过程会停止。由于我们在第k次，查找的数组下标为k，则可以说明在前k-1次中，出现数组下标为$i(1! = x), f(x) = x$。则停止时找到的，仍是我们希望找的的环。

GitHub传送门

solution one

```
class Solution {
public:
    int findDuplicate(vector<int>& nums) {
        unordered_set<int> set;
        int result = 0;
        for(auto num:nums){
            if(set.find(num)!=set.end()){
                result = num;
                break;
            }else{
                set.insert(num);
            }

        }
        return result;
    }
};
```

solution two

```
class Solution
{
  public:
    int findDuplicate(vector<int> &nums)
    {
        int low = 1;
        int high = nums.size() - 1;
        do
        {
            int count = 0;
            int mid = (low + high) / 2;
            for (auto num:nums)
            {
                if (num <= mid)
                    count++;
            }
            if (count > mid)
                high = mid;
            else
                low = mid+1;
        } while (low < high);
        return low;
    }
};
```

solution three

```
class Solution
{
  public:
    int findDuplicate(vector<int> &nums)
    {
        int slow = 0;
        int fast = 0;
        int finder = 0;
        while (true)
        {
            slow = nums[slow];
            fast = nums[nums[fast]];
            if (slow == fast)
                break;
        }
        while (true)
        {
            slow = nums[slow];
            finder = nums[finder];
            if (slow == finder)
                return slow;
        }
    }
};
```