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.

Note:

1. You **must not** modify the array (assume the array is read only).
2. You must use only constant, *O*(1) extra space.
3. Your runtime complexity should be less than `O(n^2)` .
4. There is only one duplicate number in the array, but it could be repeated more than once.

Credits:
Special thanks to @jianchao.li.fighter for adding this problem and creating all test cases.

Solution 1

The main idea is the same with problem***Linked List Cycle II***,*https://leetcode.com/problems/linked-list-cycle-ii/*. Use two pointers the fast and the slow. The fast one goes forward two steps each time, while the slow one goes only step each time. They must meet the same item when slow==fast. In fact, they meet in a circle, the duplicate number must be the entry point of the circle when visiting the array from nums[0]. Next we just need to find the entry point. We use a point(we can use the fast one before) to visit form begining with one step each time, do the same job to slow. When fast==slow, they meet at the entry point of the circle. The easy understood code is as follows.

```
int findDuplicate3(vector<int>& nums)
{
    if (nums.size() > 1)
    {
        int slow = nums[0];
        int fast = nums[nums[0]];
        while (slow != fast)
        {
            slow = nums[slow];
            fast = nums[nums[fast]];
        }

        fast = 0;
        while (fast != slow)
        {
            fast = nums[fast];
            slow = nums[slow];
        }
        return slow;
    }
    return -1;
}
```

written by echoxiaolee original link here

Solution 2

This solution is based on binary search.

At first the search space is numbers between 1 to n. Each time I select a number `mid` (which is the one in the middle) and count all the numbers equal to or less than `mid`. Then if the `count` is more than `mid`, the search space will be `[1 mid]` otherwise `[mid+1 n]`. I do this until search space is only one number.

Let's say `n=10` and I select `mid=5`. Then I count all the numbers in the array which are less than equal `mid`. If the there are more than `5` numbers that are less than `5`, then by Pigeonhole Principle (https://en.wikipedia.org/wiki/Pigeonhole_principle) one of them has occurred more than once. So I shrink the search space from `[1 10]` to `[1 5]`. Otherwise the duplicate number is in the second half so for the next step the search space would be `[6 10]`.

```
class Solution(object):
    def findDuplicate(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        low = 1
        high = len(nums)-1

        while low < high:
            mid = low+(high-low)/2
            count = 0
            for i in nums:
                if i <= mid:
                    count+=1
            if count <= mid:
                low = mid+1
            else:
                high = mid
        return low
```

There's also a better algorithm with `O(n)` time. Please read this very interesting solution here: http://keithschwarz.com/interesting/code/?dir=find-duplicate

written by mehran original link here

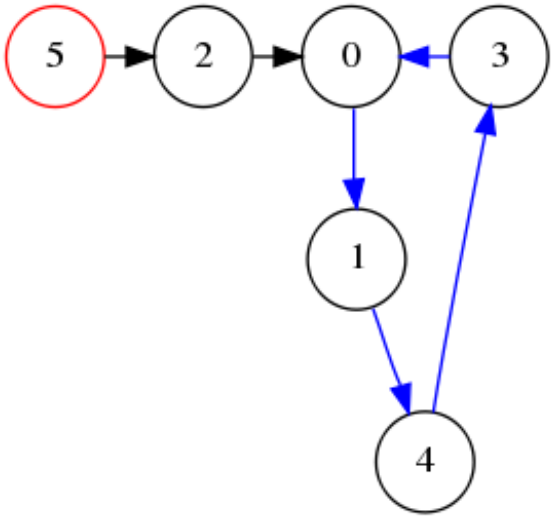
Solution 3

suppose the array is

> index: 0 1 2 3 4 5
>
> value: 2 5 1 1 4 3

first subtract 1 from each element in the array, so it is much easy to understand. use the value as pointer. the array becomes:

> index: 0 1 2 3 4 5
>
> value: 1 4 0 0 3 2



Second if the array is

> index: 0 1 2 3 4 5
>
> value: 0 1 2 4 2 3

we must choose the last element as the head of the linked list. If we choose 0, we can not detect the cycle.

Now the problem is the same as find the cycle in linkedlist!

```
public int findDuplicate(int[] nums) {
    int n = nums.length;
    for(int i=0;i<nums.length;i++) nums[i]--;
    int slow = n-1;
    int fast = n-1;
    do{
        slow = nums[slow];
        fast = nums[nums[fast]];
    }while(slow != fast);
    slow = n-1;
    while(slow != fast){
        slow = nums[slow];
        fast = nums[fast];
    }
    return slow+1;
}
```

One condition is we cannot modify the array. So the solution is

```
public int findDuplicate(int[] nums) {
    int n = nums.length;
    int slow = n;
    int fast = n;
    do{
        slow = nums[slow-1];
        fast = nums[nums[fast-1]-1];
    }while(slow != fast);
    slow = n;
    while(slow != fast){
        slow = nums[slow-1];
        fast = nums[fast-1];
    }
    return slow;
}
```

written by zq670067 original link here