LC81. Search in Rotated Sorted Array II

Suppose an array sorted in ascending order is rotated at some pivot unknown to you beforehand.

(i.e., `[0,0,1,2,2,5,6]` might become `[2,5,6,0,0,1,2]`).

You are given a target value to search. If found in the array return true, otherwise return false.

Example 1:

```
Input: nums = [2,5,6,0,0,1,2], target = 0
Output: true
```

Example 2:

```
Input: nums = [2,5,6,0,0,1,2], target = 3
Output: false
```

Follow up:
This is a follow up problem to Search in Rotated Sorted Array, where nums may contain duplicates.
Would this affect the run-time complexity? How and why?

Solution

If nums didn't contain duplicates, we only need to check two cases. We can use 3 steps 1 check to finish this problem.

1. range: `l=0,r=n-1`;
2. according to mid to updated l and r:

- first we need to check `if(nums[mid] == target) return ture`.
- then we need to find which half target is in. Because of the rotated array, we fist need to find the sorted half part by comparing the mid value and right value.
  –if(nums[mid]<nums[r]), this means the latter half is sorted. so if(nums[mid]<target<=nums[r]), update `l=mid+1`;
  otherwise target in first half, `r = mid-1`.
  – if(nums[mid]>nums[r]), this means the first half is sorted, so if(nums[l]<= target<nums[mid]), update `r = mid-1`; otherwise target is in latter half, `l = mid+1`.
  –if(nums[mid]==nums[r]), there are 2 cases.
  `1st:`
  nums[l]==nums[mid], like `[2,2,2,0,2] or [2,0,2,2,2], t=0`. In these two array, we don't know where target is only by mid,left and right value. so we can `discard left, right value` because they are unuseful for finding `t`. even if `t=2`, we still have mid value which equals left, right value, to get a correct result.
  `2nd:`
  `[3,1,2,2,2] or [1,1,2,2,2], t=1`.
  Either nums[mid]<nums[left] or nums[mid]>nums[left], t is in first half, `r= mid-1`;
  If t was in latter half, nums[mid] should = nums[l] because of the sorted array. like `[A,B,2,t,2]`.if t<2,t is the pivot, must be

`t<=2(nums[r])<=A<=B<=2nums[mid]`; if t>2, nums[r] is the pivot, must be `2(nums[r])<=A<=B<=2(nums[mid])<=t`. Thus `A=B=2`,like `[2,2,2,3,2] or [2,2,2,1,2]`. This case is exact 1st case. So we only consider the case that target is in the first half. `r = mid-1`.

3. loop condition: because we may can't find the target in nums, `while(l<=r)`.
4. check. `nums = [], or [3,1] t=1/3.`

Code

```
bool search(vector<int>& nums, int target) {
    int l = 0, r = nums.size()-1;
    while(l<=r){
        int mid = l + (r-l)/2;
        if(nums[mid] == target) return true;
        else if(nums[mid] < nums[r]){
            if(target > nums[mid] && target<= nums[r])
                l = mid + 1;
            else
                r = mid - 1;
        }
        else if(nums[mid] > nums[r]){
            if(target>= nums[l] && target < nums[mid])
                r = mid-1;
            else
                l = mid+1;
        }
        else{
            if(nums[mid] == nums[l]){
                l++;
                r--;
            }
            else
                r = mid - 1;
        }
    }
    return false;
}
```

`T(n) = O(log n), space is O(1)`

Note

1. For the 2nd case when nums[mid]=nums[r], we can't use `if((target>=nums[l] && target<nums[mid])|| (target<=nums[l]&& target>nums[mid]))` to update `r`. `E.X [3,1,2,2,2]`.
2. After getting mid, we can compare mid with left or right. However, if we compare mid with left, we need to pay more attention to two numbers case, like `[3,1], t=1`. In this case, nums[mid]==nums[l] because we use mid=l+(r-l)/2. But there will no case nums[mid]==nums[r] if nums contain no duplicates. To summarize, each time it's better to compare mid with right value.

Follow up discussion

If nums contain duplicates, the worst time complexity is `O(n)`. For instance, `[2,2,2,1,2,2,2,2,2] t=1`, then we need to move l and r for n/2 steps, so `T(n) = O(n/2) = O(n)`.

Summary

Binary Search