DFS collection

Derrick Guo

February 2020

The idea to solve the modified Binary Search problem is, well, by doing the modified binary search. There are 2 ways of doing the binary search.
1. After define l and r, set the condition of the while loop to **while l < r**, and when updating l and r, update them as, l = mid+1, r = mid. **When to use this? When there is no explicit targets.** EX: Find the peak element, or find the max in a rotated sorted array, etc. And at the end we just return nums[l].
2. **When there is an explicit target that we want to search for**, we set the while loop condition to be l <= r, and when we find the target, we break. Then when updating l and r, we set l = mid + 1, or r = mid - 1. At the end we return not found it the target is not found in the loop and we are out of the loop.

1 35. Search Insert Position

Given a sorted array and a target value, return the index if the target is found. If not, return the index where it would be if it were inserted in order.

You may assume no duplicates in the array.

Example 1:

Input: [1,3,5,6], 5 Output: 2 Example 2:

Input: [1,3,5,6], 2 Output: 1 Example 3:

Input: [1,3,5,6], 7 Output: 4 Example 4:

Input: [1,3,5,6], 0 Output: 0

idea:

Traditional binary search problem. We can use this question as the template for all modified binary search problems.

```
class Solution(object):
    def searchInsert(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        if not nums: return 0
        left, right = 0, len(nums)-1
        while left <= right:
            mid = (left+right) / 2
            if nums[mid] == target:
                return mid
            elif nums[mid] < target:
                left += 1
            else:
                right -= 1
        return left
```

2 153. Find Minimum in Rotated Sorted Array

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

(i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]).

Find the minimum element.

You may assume no duplicate exists in the array.

Example 1:

Input: [3,4,5,1,2] Output: 1 Example 2:

Input: [4,5,6,7,0,1,2] Output: 0

idea:

Traditional binary search. l = 0, r = len - 1.
Here we check if nums[mid] > r, (we need to be careful about when to use > or >=) because we know that if nums[mid] > r, then nums[mid] is guaranteed not to be the minimum we are looking for because there is something less than it. Then we are safe to say, l = mid + 1 =¿ Since we are looking for the minimum in the rotated sorted array, the minimum must be in the unsorted half.
Otherwise, the left half is the unsorted part, so we update r = mid. The reason not r = mid - 1 is that nums[mid] could be the minimum we are looking for.
The base case of the while loop is that r and l points to the same element, which makes l == r instead of l < r, therefore it breaks. So we can return either nums[l] or nums[r]. Usually just nums[l].

```
class Solution(object):
    def findMin(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums: return 0
        l, r = 0, len(nums) - 1
        while l < r:
            mid = (l + r) / 2
            if nums[mid] > nums[r]:
                l = mid + 1
            else:
                r = mid
        return nums[l]
```

3 154. Find Minimum in Rotated Sorted Array II

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

(i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]).

Find the minimum element.

The array may contain duplicates.

Example 1:

Input: [1,3,5] Output: 1 Example 2:

Input: [2,2,2,0,1] Output: 0

idea:

If there is duplicates in the array, there is no way we can tell if the min is in the left half or the right half, because it could happen that nums[l] == nums[mid] == nums[r] and the minimum hides in the middle (ex: [2,2,2,2,2,2,2,1,2]). Therefore, it's easier to use recursion and find the minimum of both of the subarrays and choose the min between them.

```python
class Solution(object):
    def findMin(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums: return 0
        return self.find(nums, 0, len(nums)-1)

    def find(self, nums, l, r):
        if l == r:
            return nums[l]
        if nums[l] < nums[r]: # sorted
            return nums[l]

        mid = (l+r) / 2
        return min(self.find(nums, l, mid), self.find(nums, mid+1, r))
```

4 33. Search in Rotated Sorted Array

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

(i.e., [0,1,2,4,5,6,7] might become [4,5,6,7,0,1,2]).

You are given a target value to search. If found in the array return its index, otherwise return -1.

You may assume no duplicate exists in the array.

Your algorithm's runtime complexity must be in the order of O(log n).

Example 1:

Input: nums = [4,5,6,7,0,1,2], target = 0 Output: 4 Example 2:

Input: nums = [4,5,6,7,0,1,2], target = 3 Output: -1

idea:

This is a good example of the basic search in rotated sorted array problems. First we define mid = (l+r)/2, then check if nums[mid] is the target. If so, return the index, otherwise check which half of the array is sorted. For example, If the left half of the array is sorted (nums[l] < nums[mid]), and the target is between nums[l] and nums[mid] so we update r = mid - 1, (the reason we don't update r = mid is that we check if mid is the target each time, so if the code executes until this point then we know nums[mid] is not the target.) Otherwise, we know that the target is not in the left half so we update l = mid + 1. Similary, in the else statement, we check if the target is in the right half, if so we update l = mid + 1, otherwise r = mid - 1. We can't replace else with elif nums[r] ¿ nums[mid], because else needs to cover the situation that the target is not in the array.

We return -1 at the end, indicating we can't find such target.

```python
class Solution(object):
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: int
        """
        if not nums: return -1
        l, r = 0, len(nums) - 1
        while l <= r:
            mid = (l+r) / 2
            if nums[mid] == target:
                return mid
            elif nums[mid] <= nums[r]:
                if nums[mid] <= target <= nums[r]:
                    l = mid + 1
                else:
                    r = mid - 1
            elif nums[mid] >= nums[l]:
                if nums[l] <= target <= nums[mid]:
                    r = mid - 1
                else:
                    l = mid + 1
        return -1
```

5 81. 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

idea:

When there is duplicates, there is change that mid doesn't give us any useful information. But in some cases it does provide information like the last question. So we need to do everything in the previous question, but with a little enhancement: we check if nums[l] == nums[mid]. If so, that means we have no idea of what's going on in the left half, so we simple increase l by 1. This will also increase mid by 1, and by doing this step by step we can access the hidden information in the left half of the array. Otherwise, if nums[l] ¡ nums[mid], it still means that the left half is sorted, we check if the target is in there and update l and r correspondingly; and else, nums[l] ¿ nums[mid], the right half is sorted and we do the same thing.

```
class Solution(object):
    def search(self, nums, target):
        """
        :type nums: List[int]
        :type target: int
        :rtype: bool
        """
        if not nums: return False
        l, r = 0, len(nums)-1
        while l <= r:
            mid = (l+r) / 2
            if nums[mid] == target:
                return True
            elif nums[l] < nums[mid]:
                if nums[l] <= target <= nums[mid]:
                    r = mid-1
                else:
                    l = mid+1
            elif nums[l] == nums[mid]:
                l += 1
            elif nums[r] >= nums[mid]:
                if nums[mid] <= target <= nums[r]:
                    l = mid+1
                else:
                    r = mid-1

        return False
```

6 162. Find Peak Element

A peak element is an element that is greater than its neighbors.

Given an input array nums, where nums[i] nums[i+1], find a peak element and return its index.

The array may contain multiple peaks, in that case return the index to any one of the peaks is fine.

You may imagine that nums[-1] = nums[n] = -.

Example 1:

Input: nums = [1,2,3,1] Output: 2 Explanation: 3 is a peak element and your function should return the index number 2. Example 2:

Input: nums = [1,2,1,3,5,6,4] Output: 1 or 5 Explanation: Your function can return either index number 1 where the peak element is 2, or index number 5 where the peak element is 6.

idea:

This requires a bit more modifications to the binary search. Basically what we are looking for is a trend, since we are looking for a peak value. Then instead of finding only 1 mid, we use mid2 = mid1+1, which is the index of the element to the right of nums[mid]. Then we compare the two. If nums[mid] ¿ nums[mid2], then we know that there is a downward trend from mid1 to mid2, then the peak value is more likely towards left, which is mid1, then we set r = mid1. Otherwise, there is an upward trend from mid1 to mid2, which means mid2 is more likely to lead us to a peak value (or itself is one), so we set l = mid2. The break point of the while loop is when l == r, so we return l at the end.

```
class Solution(object):
    def findPeakElement(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums: return 0
        l, r = 0, len(nums) - 1
        while l < r:
            mid1 = (l + r) / 2
            mid2 = mid1 + 1
            if nums[mid1] > nums[mid2]:
                r = mid1
            else:
                l = mid2
        return l
```

7 222. Count Complete Tree Nodes

Given a complete binary tree, count the number of nodes.

Note:

Definition of a complete binary tree from Wikipedia: In a complete binary tree every level, except possibly the last, is completely filled, and all nodes in the last level are as far left as possible. It can have between 1 and 2h nodes inclusive at the last level h.

Example:

Input:

1
/ \
2 3
/ \/
4 5 6

Output: 6

idea:

Modified binary search are also applied in BST a lot.
This question can be done by a simple 2 lines DFS, where you return 0 if the node you are on is null, and 1 + dfs(left) + dfs(right) otherwise. But this actually requires looping through all of the tree which is O(n). We can do better by doing binary search since we are given a complete tree.

By the definition of a complete tree, we know all the nodes are as far left as possible. Then we can count the length of the left and right subtree. If the left subtree is longer, then we know the right subtree is complete. Then we can calculate the number of the nodes by 2 ∗∗ (the length of the right subtree) plus countNode(left subtree). Otherwise the length of left subtree is the same as the length of the right subtree. Then we know the left subtree is complete, so we do 2 ∗∗ (length of left subtree) + countNode(right subtree). Each pass of getLength call is O(logn), and we traverse the tree from top to bottom and called getLength O(logn) times, so the whole run time of this is O((logn)**2)).

Both solutions are listed below.

```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def countNodes(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        if root == None: return 0
        return self.dfs(root)

    def dfs(self, root):
        if not root: return 0
        return 1 + self.dfs(root.left) + self.dfs(root.right)
```

```python
# Definition for a binary tree node.
# class TreeNode(object):
#     def __init__(self, x):
#         self.val = x
#         self.left = None
#         self.right = None

class Solution(object):
    def countNodes(self, root):
        """
        :type root: TreeNode
        :rtype: int
        """
        if root == None: return 0
        leftLen = self.getLength(root.left)
        rightLen = self.getLength(root.right)
        if rightLen == leftLen:
            # left subtree is complete
            return 2 ** leftLen + self.countNodes(root.right)
        else:
            # left len > rightlen, right subtree is complete
            return 2 ** rightLen + self.countNodes(root.left)


    def getLength(self, root):
        if not root: return 0
        return 1 + self.getLength(root.left)
```

8 275. H-Index II

Given an array of citations sorted in ascending order (each citation is a non-negative integer) of a researcher, write a function to compute the researcher's h-index.

According to the definition of h-index on Wikipedia: "A scientist has index h if h of his/her N papers have at least h citations each, and the other N h papers have no more than h citations each."

Example:

Input: citations = [0,1,3,5,6] Output: 3 Explanation: [0,1,3,5,6] means the researcher has 5 papers in total and each of them had received 0, 1, 3, 5, 6 citations respectively. Since the researcher has 3 papers with at least 3 citations each and the remaining two with no more than 3 citations each, her h-index is 3. Note:

If there are several possible values for h, the maximum one is taken as the h-index.

idea:
The definition of H-index is citations[idx] == len(citations) - idx, so the len(citations) - idx here is what we need to find (assuming the array is ascendingly sorted). Then the question is similar to search insert position, we do binary search on the sorted array. Instead of returning left at the end, we return len(citations) - left because the h-index are found from the right of the array.

```python
class Solution(object):
    def hIndex(self, citations):
        """
        :type citations: List[int]
        :rtype: int
        """
        if not citations: return 0
        l, r = 0, len(citations)-1
        while l <= r:
            mid = (l+r) / 2
            if citations[mid] == len(citations) - mid:
                return citations[mid]
            if citations[mid] > len(citations) - mid:
                r = mid - 1
            else:
                l = mid + 1
        return len(citations)-l
```