Search in Rotated Sorted Array II
/ When there are duplicates, the worst case is O(n). Could we do better?

When there are duplicates, the worst case is O(n). Could we do better?

28

baojialiang
Reputation: 28

Since we will have some duplicate elements in this problem, it is a little tricky because sometimes we cannot decide whether to go to the left side or right side. So for this condition, I have to probe both left and right side simultaneously to decide which side we need to find the number. Only in this condition, the time complexity may be *O(n)*. The rest conditions are always *O(log n)*.

For example:

input: `113111111111`, Looking for *target* `3`.

Is my solution correct? My code is as followed:

```
public class Solution {
    public boolean search(int[] A, int target) {
        // IMPORTANT: Please reset any member data you declared, as
        // the same Solution instance will be reused for each test case.
```

```
        int i = 0;
        int j = A.length - 1;
        while(i <= j){
            int mid = (i + j) / 2;
            if(A[mid] == target)
                return true;
            else if(A[mid] < A[i]){
                if(target > A[j])
                    j = mid - 1;
                else if(target < A[mid])
                    j = mid - 1;
                else
                    i = mid + 1;
            }else if(A[mid] > A[i]){
                if(target < A[mid] && target >= A[i])
                    j = mid - 1;
                else
                    i = mid + 1;
            }else{ // A[mid] == A[i]
                if(A[mid] != A[j])
                    i = mid + 1;
                else{
                    boolean flag = true;
                    for(int k = 1; mid - k >= i && mid + k <= j; k++){
                        if(A[mid] != A[mid - k]){
                            j = mid - k;
                            flag = false;
                            break;
                        }else if(A[mid] != A[mid + k]){
                            i = mid + k;
                            flag = false;
                            break;
```

216

1337c0d3r
Reputation: 1576

Yes, when there could be duplicates in the array, the worst case is *O(n)*.

To explain why, consider this sorted array `1111115`, which is rotated to `1151111`.

Assume `left = 0` and `mid = 3`, and the *target* we want to search for is `5`. Therefore, the condition `A[left] == A[mid]` holds true, which leaves us with only *two* possibilities:

1. All numbers between `A[left]` and `A[right]` are all 1's.
2. Different numbers (including our *target*) may exist between `A[left]` and `A[right]`.

As we cannot determine which of the above is true, the best we can do is to move `left` one step to the right and repeat the process again. Therefore, we are able to construct a worst case input which runs in *O(n)*, for example: the input `11111111...115`.

Below is a pretty concise code (thanks to **bridger**) for your reference which I found from the old discuss.

```
bool search(int A[], int n, int key) {
    int l = 0, r = n - 1;
    while (l <= r) {
        int m = l + (r - l)/2;
        if (A[m] == key) return true; //return m in Search in Rotated Array I
        if (A[l] < A[m]) { //left half is sorted
            if (A[l] <= key && key < A[m])
                r = m - 1;
            else
                l = m + 1;
        } else if (A[l] > A[m]) { //right half is sorted
            if (A[m] < key && key <= A[r])
                l = m + 1;
            else
```

```
                r = m - 1;
            } else l++;
        }
        return false;
    }
```

0

B

baojialiang @1337c0d3r
Reputation: 28

That is great! The code you listed above is so concise! :)

1

S

sibinnuosha @1337c0d3r
Reputation: 1

I think you should check if (A[l] == target) before l++

0

M

magicknife @1337c0d3r
Reputation: 70

sibinnuosha is right. Also, you could check both A[l] and A[r] for equality and move both ends (i.e. l++ r--)

0

I

its_dark @1337c0d3r
Reputation: 10

No need to check

2

its_dark

Reputation: 10

A bit different version of the code posted by 1337c0d3r:

```
bool search(int A[], int N, int key) {
       int L = 0;
 int R = N - 1;

 while (L <= R) {
   // Avoid overflow, same as M=(L+R)/2
   int M = L + ((R - L) / 2);
   if (A[M] == key) return true;

   // the bottom half is sorted
label:
   if (A[L] <= A[M]) {
     if(A[L]==A[M] && L !=M){
         while(L != M && A[L]==A[M])L++;
         goto label;
     }

     if (A[L] <= key && key < A[M])
       R = M - 1;
     else
       L = M + 1;
   }
   // the upper half is sorted
   else {
     if (A[M] < key && key <= A[R])
       L = M + 1;
```

```
        else
          R = M - 1;
      }
    }
    return false;
}
```

0

sweetsweat @1337c0d3r
Reputation: 0

I don't think this answer is optimal. For example, for input 2222222222222221 and target 1, the algorithm above would be O(n), but it actually can be done in O(logn). Basically, whenever you see A[m] == A[l] && A[m] != A[h], you know A[l] to A[m] must be identical, so you only need to search the right half. So is the case of A[m] != A[l] && A[m] == A[h]. The only case you need to search both half is when A[m] == A[l] == A[h].

0

Szilard @1337c0d3r
Reputation: 4

Actually "optimality" refers to the asymptotic complexity of the algorithm. One might argue that just traversing the whole array is O(N) and because we can't really do better in the worst case, it is optimal. But we can always make particular cases work better and in practice, we usually should do so knowing what the input data is. Anyway, good catch :), indeed we can make the algorithm faster for certain cases. like you mentioned, the O(n), if coded correclty should occur only in case A[m] == A[l] == A[h]

0

arthur1026
Reputation: 0

This post is deleted!

2

lurklurk
Reputation: 93

A different method. If we see A[l]==A[m] or A[m]==A[r], we can increase the value of l, or decrease value of r until we get a different value of A[l] or A[r]

```
public boolean searchII(int[] A, int target) {
        if (A == null) return false;
        int start = 0, end = A.length -1;
        while(start <= end){
                int mid = (start+end)/2;
                if (target == A[mid])
                        return true;
                boolean duplicated = false;
                while(start<=mid && A[start]==A[mid]){
                        start++;
                        duplicated = true;
                }
                while(end >= mid && A[mid] == A[end]){
                        end--;
                        duplicated = true;
                }

                if (duplicated)
                        continue;

                if (A[start] < A[mid]){
                                if (target >= A[start] && target < A[mid])
                                        end = mid - 1;
                                else
```

```
                    start = mid + 1;
            } else if (target > A[mid] && target <= A[end])
                start = mid + 1;
            else
                end = mid - 1;
        }
        return false;
}
```

2

zs1350

Reputation: 3

I would like to add this as comment of the best answer, but I don't know how to add code there.

Based on 1337c0d3r's idea of the best answer, I deleted duplicate numbers from both sides to the middle until there was a different number on any side, like from (111111112111) to (111111112), then if the number in the middle equal to first or last number, then dimiss the whole half the array, now the array became (12). At last, implement the general idea of this question.
This idea may reduce some running time, but code not seems concise enough.

From the tests' running time, this code used 24ms for all test cases.

```
class Solution {
public:
    bool search(int A[], int n, int target) {
        if(n == 0)
            return false;
        int left = 0, right = n-1;
        while(left <= right)
        {
            int mid = (left+right)/2;
```

```
            if(A[mid] == target)
                return true;
            else
            {
                if(A[left] == A[mid] && A[mid] == A[right])
                {
                    left++;
                    right--;
                }
                else if(A[left] == A[mid])
                    left = mid + 1;
                else if(A[mid] == A[right])
                    right = mid;
                else if(A[mid] > A[left])
                {
                    if(target >= A[left] && target <= A[mid])
                        right = mid;
                    else
                        left = mid + 1;
                }
                else
                {
                    if(target >= A[mid] && target <= A[right])
                        left = mid + 1;
                    else
                        right = mid;
                }
```

0

jixiangchn

Reputation: ★ -25

Everybody should read this solution. Although it has less votes than the following "concise" solution, it is better and faster than that solution in A[m] == A[i] == A[j] case. Moreover, when A[m] == A[i] == A[j], I simply traverse all the elems between i and j, but his method is better.

0

steven10 @1337c0d3r
Reputation: 18

It's a very good solution. Can anybody explain why we can only use left++ instead of right--directly, not by checking both? Thanks in advance.

0

cherler @1337c0d3r
Reputation: 0

It is the same time complexity as the Linear traversal method .Not good enough!

0

Meowgadeth @1337c0d3r
Reputation: 25

Either left++ or right-- works. We will have two possibility only when duplicated numbers are at start and end, so shifting either side works

0

EXLsunshine @1337c0d3r
Reputation: 55

I think left++ works find but right-- does not.
Because your condition is: A[l] >,<,==,A[m], so we should use left++ here.

If the condition is: A[r] ><== A[m], then we should use right--.

Is that right?

0

annieqt @1337c0d3r
Reputation: 270

@EXLsunshine I agree with you.

0

736732519 @1337c0d3r
Reputation: 0

This post is deleted!

0

Meowgadeth @1337c0d3r
Reputation: 25

Here's my answer, you can try change end-- to start++ and they both work
https://leetcode.com/discuss/60618/neat-java-solution-using-binary-search

0

RainbowSecret @1337c0d3r
Reputation: 722

clear and nice implementation

BINARY-SEARCH 566 TIME-COMPLEXITY 633

28 POSTS

14630 VIEWS

Log in to reply