Previous (/articles/binary-tree-inorder-traversal/) Next (/articles/palindrome-number/)

4. Median of Two Sorted Arrays (/problems/median-of-two-sorted-arrays/)

Aug. 30, 2017 | 141.4K views

There are two sorted arrays **nums1** and **nums2** of size m and n respectively.

Find the median of the two sorted arrays. The overall run time complexity should be O(log (m+n)).

Example 1:

```
nums1 = [1, 3]
nums2 = [2]

The median is 2.0
```

Example 2:

```
nums1 = [1, 2]
nums2 = [3, 4]

The median is (2 + 3)/2 = 2.5
```

Solution

Approach 1: Recursive Approach

To solve this problem, we need to understand "What is the use of median". In statistics, the median is used for:

> Dividing a set into two equal length subsets, that one subset is always greater than the other.

If we understand the use of median for dividing, we are very close to the answer.

First let's cut A into two parts at a random position i:

```
          left_A             |        right_A
    A[0], A[1], ..., A[i-1]  |  A[i], A[i+1], ..., A[m-1]
```

Since A has m elements, so there are $m+1$ kinds of cutting ($i = 0 \sim m$).

And we know:

> $\text{len}(\text{left\_A}) = i, \text{len}(\text{right\_A}) = m - i.$
>
> Note: when $i = 0$, left_A is empty, and when $i = m$, right_A is empty.

With the same way, cut B into two parts at a random position j:

```
          left_B             |          right_B
    B[0], B[1], ..., B[j-1]  |  B[j], B[j+1], ..., B[n-1]
```

Put $\mathrm{left\_A}$ and $\mathrm{left\_B}$ into one set, and put $\mathrm{right\_A}$ and $\mathrm{right\_B}$ into another set. Let's name them $\mathrm{left\_part}$ and $\mathrm{right\_part}$:

```
          left_part          |          right_part
    A[0], A[1], ..., A[i-1]  |  A[i], A[i+1], ..., A[m-1]
    B[0], B[1], ..., B[j-1]  |  B[j], B[j+1], ..., B[n-1]
```

If we can ensure:

> 1. $\mathrm{len(left\_part)} = \mathrm{len(right\_part)}$
> 2. $\max(\mathrm{left\_part}) \le \min(\mathrm{right\_part})$

then we divide all elements in $\{\mathrm{A}, \mathrm{B}\}$ into two parts with equal length, and one part is always greater than the other. Then

$$\mathrm{median} = \frac{\max(\mathrm{left\_part}) + \min(\mathrm{right\_part})}{2}$$

To ensure these two conditions, we just need to ensure:

> 1. $i + j = m - i + n - j$ (or: $m - i + n - j + 1$)
>   if $n \ge m$, we just need to set: $\mathrm{i = 0 \sim m,\ j = \frac{m+n+1}{2} - i}$
> 2. $\mathrm{B}[j-1] \le \mathrm{A}[i]$ and $\mathrm{A}[i-1] \le \mathrm{B}[j]$

ps.1 For simplicity, I presume $\mathrm{A}[i-1], \mathrm{B}[j-1], \mathrm{A}[i], \mathrm{B}[j]$ are always valid even if $i = 0$, $i = m$, $j = 0$, or $j = n$. I will talk about how to deal with these edge values at last.

ps.2 Why $n \ge m$? Because I have to make sure j is non-negative since $0 \le i \le m$ and $j = \frac{m+n+1}{2} - i$. If $n < m$, then j may be negative, that will lead to wrong result.

So, all we need to do is:

> Searching i in $[0, m]$, to find an object i such that:
>
> $\mathrm{B}[j-1] \le \mathrm{A}[i]$ and $\mathrm{A}[i-1] \le \mathrm{B}[j]$, where $j = \frac{m+n+1}{2} - i$

And we can do a binary search following steps described below:

1. Set $\mathrm{imin} = 0$, $\mathrm{imax} = m$, then start searching in $[\mathrm{imin}, \mathrm{imax}]$
2. Set $i = \frac{\mathrm{imin}+\mathrm{imax}}{2}$, $j = \frac{m+n+1}{2} - i$
3. Now we have $\mathrm{len(left\_part)} = \mathrm{len(right\_part)}$. And there are only 3 situations that we may encounter:
   - $\mathrm{B}[j-1] \le \mathrm{A}[i]$ and $\mathrm{A}[i-1] \le \mathrm{B}[j]$
     Means we have found the object i, so stop searching.
   - $\mathrm{B}[j-1] > \mathrm{A}[i]$
     Means $\mathrm{A}[i]$ is too small. We must adjust i to get $\mathrm{B}[j-1] \le \mathrm{A}[i]$.
     Can we increase i?
       Yes. Because when i is increased, j will be decreased.
       So $\mathrm{B}[j-1]$ is decreased and $\mathrm{A}[i]$ is increased, and $\mathrm{B}[j-1] \le \mathrm{A}[i]$ may be satisfied.
     Can we decrease i?
       No! Because when i is decreased, j will be increased.
       So $\mathrm{B}[j-1]$ is increased and $\mathrm{A}[i]$ is decreased, and $\mathrm{B}[j-1] \le \mathrm{A}[i]$ will

be never satisfied.
So we must increase i. That is, we must adjust the searching range to $[i+1, \text{imax}]$.
So, set $\text{imin} = i+1$, and goto 2.

- $\text{A}[i-1] > \text{B}[j]$:
  Means $\text{A}[i-1]$ is too big. And we must decrease i to get $\text{A}[i-1] \le \text{B}[j]$.
  That is, we must adjust the searching range to $[\text{imin}, i-1]$.
  So, set $\text{imax} = i-1$, and goto 2.

When the object i is found, the median is:

> $\max(\text{A}[i-1], \text{B}[j-1]),$ when $m+n$ is odd
>
> $\frac{\max(\text{A}[i-1],\text{B}[j-1]) + \min(\text{A}[i],\text{B}[j])}{2},$ when $m+n$ is even

Now let's consider the edges values $i=0, i=m, j=0, j=n$ where $\text{A}[i-1], \text{B}[j-1], \text{A}[i], \text{B}[j]$ may not exist. Actually this situation is easier than you think.

What we need to do is ensuring that $\max(\text{left\_part}) \le \min(\text{right\_part})$. So, if i and j are not edges values (means $\text{A}[i-1], \text{B}[j-1], \text{A}[i], \text{B}[j]$ all exist), then we must check both $\text{B}[j-1] \le \text{A}[i]$ and $\text{A}[i-1] \le \text{B}[j]$. But if some of $\text{A}[i-1], \text{B}[j-1], \text{A}[i], \text{B}[j]$ don't exist, then we don't need to check one (or both) of these two conditions. For example, if $i=0$, then $\text{A}[i-1]$ doesn't exist, then we don't need to check $\text{A}[i-1] \le \text{B}[j]$. So, what we need to do is:

> Searching i in $[0, m]$, to find an object i such that:
>
> $(j=0$ or $i=m$ or $\text{B}[j-1] \le \text{A}[i])$ and
> $(i=0$ or $j=n$ or $\text{A}[i-1] \le \text{B}[j])$, where $j = \frac{m+n+1}{2} - i$

And in a searching loop, we will encounter only three situations:

> 1. $(j=0$ or $i=m$ or $\text{B}[j-1] \le \text{A}[i])$ and
> $(i=0$ or $j=n$ or $\text{A}[i-1] \le \text{B}[j])$
>   Means i is perfect, we can stop searching.
> 2. $j>0$ and $i<m$ and $\text{B}[j-1] > \text{A}[i]$
>   Means i is too small, we must increase it.
> 3. $i>0$ and $j<n$ and $\text{A}[i-1] > \text{B}[j]$
>   Means i is too big, we must decrease it.

Thanks to @Quentin.chen (https://leetcode.com/Quentin.chen) for pointing out that: $\text{i} < \text{m} \implies \text{j} > 0$ and $\text{i} > 0 \implies \text{j} < \text{n}$. Because:

> $\text{m} \le \text{n},\ \text{i} < \text{m} \implies \text{j} = \frac{\text{m+n+1}}{2} - \text{i} > \frac{\text{m+n+1}}{2} - \text{m} \ge \frac{2\text{m}+1}{2} - \text{m} \ge 0$
>
> $\text{m} \le \text{n},\ \text{i} > 0 \implies \text{j} = \frac{\text{m+n+1}}{2} - \text{i} < \frac{\text{m+n+1}}{2} \le \frac{2\text{n}+1}{2} \le \text{n}$

So in situation 2. and 3. , we don't need to check whether $j > 0$ and whether $j < n$.

Java Python Copy

```
        }
        int iMin = 0, iMax = m, halfLen = (m + n + 1) / 2;
        while (iMin <= iMax) {
            int i = (iMin + iMax) / 2;
            int j = halfLen - i;
            if (i < iMax && B[j-1] > A[i]){
                iMin = iMin + 1; // i is too small
            }
            else if (i > iMin && A[i-1] > B[j]) {
                iMax = iMax - 1; // i is too big
            }
            else { // i is perfect
                int maxLeft = 0;
                if (i == 0) { maxLeft = B[j-1]; }
                else if (j == 0) { maxLeft = A[i-1]; }
                else { maxLeft = Math.max(A[i-1], B[j-1]); }
                if ( (m + n) % 2 == 1 ) { return maxLeft; }

                int minRight = 0;
                if (i == m) { minRight = B[j]; }
                else if (j == n) { minRight = A[i]; }
                else { minRight = Math.min(B[j], A[i]); }

                return (maxLeft + minRight) / 2.0;
            }
        }
        return 0.0;
```

Complexity Analysis

- Time complexity: $O\big(\log\big(\min(m,n)\big)\big)$.
  At first, the searching range is $[0, m]$. And the length of this searching range will be reduced by half after each loop. So, we only need $\log(m)$ loops. Since we do constant operations in each loop, so the time complexity is $O\big(\log(m)\big)$. Since $m \le n$, so the time complexity is $O\big(\log\big(\min(m,n)\big)\big)$.
- Space complexity: $O(1)$.
  We only need constant memory to store 9 local variables, so the space complexity is $O(1)$.

Rate this article:

(/ratings/107/232/?return=/articles/median-of-two-sorted-arrays/) (/ratings/107/232/?return=/a

Previous (/articles/binary-tree-inorder-traversal/)

Next (/articles/palindrome-number/)

Comments: 168

Sort By

Type comment here... (Markdown is supported)

Preview Post

LazyWolfLin (/lazywolflin) ★ 0 39 minutes ago

I submit two solution, one is O(min(m, n)) and the other one is O(log(min(m,n))). But they get the same runtime. I think those test cases may too little and too weak.

0 Share Reply

LazyWolfLin (/lazywolflin) ★ 0 an hour ago

The implementation is only O(min(m,n)) both Java and Python.

0 Share Reply

ryanleitaiwan (/ryanleitaiwan) ★ 3 3 days ago

The proof for eliminating condition is slightly wrong! Obviously, (2n+1)/2 <= n is buggy. Variable i should have been kept for later. You can instead say: j = (m+n+1)/2 - i <= (2n+1)/2 - i = n + 1/2 - i < n because i is an integer greater than 0.

Also in the previous proof, I think you can remove the equal sign at (2m+1)/2 - m >= 0

Read More

1 Share Reply

CodeP (/codep) ★ 13 July 11, 2018 5:51 AM