

[每日一题]4. Median of Two Sorted Arrays

原创 2017-05-03 东东 每日一题算法题

题目描述

难度 Hard

主题 分治算法

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]`

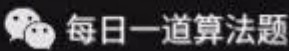
The median is $(2 + 3)/2 = 2.5$

解题方案

暴力法

将两个数组合并后，求其中位数。

```
1 class Solution {
2 public:
3     double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
4         if(nums1.size() + nums2.size() == 0) return 0;
5         for(auto i : nums2){
6             nums1.push_back(i);
7         }
8         sort(nums1.begin(),nums1.end());
9         if(nums1.size() % 2){
10             return nums1[nums1.size()/2];
11         }
12         else{
13             return (double)(nums1[nums1.size()/2-1]+nums1[nums1.size()/2])/2;
14         }
15     }
16 };
```



Text

分治算法

假设两个数组的长度分别是 m 和 n ，它们的数组长度之和为 k 。本题的意思就是在合并的数组中找 $k/2$ 个数。

如果 k 是偶数，那么就是 $k/2$ ；如果 k 是奇数，那么就是 $k/2$ 与 $k/2 + 1$ 的均值。

先来说说 k 是奇数的情况， k 是偶数的情况同理。

如果 a 数组里里面第 $k/2$ 个数比 b 数组里面的大，这说明我们要找的数一定不在 b 的前 $k/2$ 个元素里。我们使用反证法证明。假设 a 、 b 的长度都是 1000，那么 $k = 2000$ ， $k/2 = 1000$ ， a 的第 500 个数比 b 的第 500 个数大。如果我们要找的数在 b 的前 500 个数里，那么 a 数组的前 500 个数一定在小于 b 数组的第 500 个数，因为只有这样才能凑齐 1000 个数。跟我们的假设 a 的第 500 个数大于 b 的第 500 个数矛盾。同理可以求证，我们要找的数，也不可能在 a 的后 $k/2$ 个数里。

还需要考虑一个特殊情况，当 $k/2$ 已经大于 a 的长度，说明我们要找的数字一定不在 a 里， b 同理。

以上分析由二群 @DSC 提供

Submitted Code: 0 minutes ago

Language: java

Edit C

```
1 public class Solution {
2     public double findMedianSortedArrays(int[] nums1, int[] nums2) {
3         if (nums1 == null || nums1.length == 0) {
4             return median(nums2);
5         } else if (nums2 == null || nums2.length == 0) {
6             return median(nums1);
7         }
8         int len = nums1.length + nums2.length;
9         if (len % 2 == 1) {
10             return find(nums1, 0, nums2, 0, len / 2 + 1) + 0.0;
11         }
12         return (find(nums1, 0, nums2, 0, len / 2) + find(nums1, 0, nums2, 0, len / 2 + 1)) / 2.0;
13     }
14
15     private int find(int[] a, int a_left, int[] b, int b_left, int k) {
16         if (a_left >= a.length) {
17             return b[b_left + k - 1];
18         }
19         if (b_left >= b.length) {
20             return a[a_left + k - 1];
21         }
22         if (k == 1) {
23             return Math.min(a[a_left], b[b_left]);
24         }
25         int a_mid = k / 2 + a_left - 1 < a.length ? a[a_left + k / 2 - 1] : Integer.MAX_VALUE;
26         int b_mid = k / 2 + b_left - 1 < b.length ? b[b_left + k / 2 - 1] : Integer.MAX_VALUE;
27         if (a_mid > b_mid) {
28             b_left = b_left + k / 2;
29         } else {
30             a_left = a_left + k / 2;
31         }
32         return find(a, a_left, b, b_left, k - k / 2);
33     }
34
35     private double median(int[] num) {
36         if (num == null || num.length == 0) {
37             return 0.0;
38         } else if (num.length % 2 == 1) {
39             return num[num.length / 2];
40         }
41         return (num[num.length / 2 - 1] + num[num.length / 2]) / 2.0;
42     }
43 }
```

 每日一题算法题[Back to problem](#)

```

1  using namespace std;
2
3  class Solution {
4  public:
5      double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
6          int length_1 = nums1.size();
7          int length_2 = nums2.size();
8          int length = length_1 + length_2;
9          if (length % 2 == 0)
10         {
11             double result_1 = find_kth_number(nums1, nums2, length / 2);
12             double result_2 = find_kth_number(nums1, nums2, length / 2 + 1);
13             return (result_1 + result_2) / 2;
14         }
15         return find_kth_number(nums1, nums2, length / 2 + 1);
16     }
17     double find_kth_number(vector<int>& nums1, vector<int>& nums2, int k)
18     {
19         int length_1 = nums1.size();
20         int length_2 = nums2.size();
21         if (length_1 == 0)
22             return nums2[k - 1];
23         if (length_2 == 0)
24             return nums1[k - 1];
25         int mid_1 = (length_1 - 1) / 2; // mid is the middle subscript in array
26         int mid_2 = (length_2 - 1) / 2; // k is the actual position in array. NOTICE!
27         if (nums1[mid_1] <= nums2[mid_2])
28             if (mid_1 + mid_2 + 1 >= k)
29             {
30                 vector<int> new_nums2;
31                 vector<int>::iterator it2 = nums2.begin();
32                 new_nums2.assign(it2, it2 + mid_2);
33                 return find_kth_number(nums1, new_nums2, k);
34             }
35             else
36             {
37                 vector<int> new_nums1;
38                 vector<int>::iterator it1 = nums1.begin();
39                 new_nums1.assign(it1 + mid_1 + 1, nums1.end());
40                 return find_kth_number(new_nums1, nums2, k - mid_1 - 1);
41             }
42         else if (nums2[mid_2] <= nums1[mid_1])
43             if (mid_1 + mid_2 + 1 >= k)
44             {
45                 vector<int> new_nums1;
46                 vector<int>::iterator it1 = nums1.begin();
47                 new_nums1.assign(it1, it1 + mid_1);
48                 return find_kth_number(nums2, new_nums1, k);
49             }
50             else
51             {
52                 vector<int> new_nums2;
53                 vector<int>::iterator it2 = nums2.begin();
54                 new_nums2.assign(it2 + mid_2 + 1, nums2.end());
55                 return find_kth_number(new_nums2, nums1, k - mid_2 - 1);
56             }
57     }
58 };

```

 每日一题算法题

二分法

假设数组的长度分别是 m 和 n ，把数组各自分成两个部分，他们的下标分别是 i 和 j ，那么划分后的数组分别是 $a[0] \dots a[i-1]$ 和 $a[i] \dots a[m-1]$ ， $b[0] \dots b[j-1]$ 和 $b[j] \dots b[n-1]$ 。在分割数组的时候，我们可以保持 i 自有变动， j 的值始终 $i + j = (m + n + 1) / 2$ 计算出。那么如果找到第一个 i ，使得分割后的数组满足： $a[i-1] < b[j]$ 并且 $b[j-1] < a[i]$ 的话，如果我们把 $a[0] \dots a[i-1]$ 和 $b[0] \dots b[j-1]$ 合并成在一起(称为左边)，把 $a[i] \dots a[m-1]$ 和 $b[j] \dots b[n-1]$ 合并在一起(称为右边)，可以观察出左右两边的数组的元素个数几乎一样多(左边会比右边多一个)，并且左边的最大元素会比右边的最小元素小，符合 Median 的定义。那么整体的 median 数就一定在左右两边的边界上了。

以上思路由第6群的 @晓萌 提供

```

/*
 *          left      |      right
 * nums1[0]...nums1[i-1] | nums1[i]...nums1[m-1]
 * nums2[0]...nums2[j-1] | nums2[j]...nums2[n-1]
 *
 * Goal: find an i such that nums1[i-1] < nums2[j] && nums2[j-1] < nums1[i] where j=(m+n+1)/2 - i
 * Once i is found, medium = (left_max+right_min)/2 iff (m+n)%2 == 0
 * or medium = (left_max) iff (m+n)%2 == 1
 *
 * left_max = max(nums1[i-1], nums2[j-1])
 * right_min = min(nums1[i], nums2[j])
 *
 * Question: how to find i?
 *
 * Given an i, if(nums1[i-1] > nums2[j]) => need to decrease i so that j would increase => get larger nums2[j]
 * if(nums1[i] < nums2[j-1]) => need to increase i so that j would decrease => get smaller nums2[j-1]
 * Use the idea of binary search! => time complexity is O(log(min(m,n)))
 *
 * corner cases:
 * how to handle i == 0, i == m, j == 0, j == n?
 * Assume m <= n, otherwise we just switch the arrays, i.e. name the original nums1 as nums2, and name the original
 * nums2 as nums1
 * Also assume m > 0, because we could easily take care of m = 0 separately.
 * if i == 0 || j == n, no need to compare nums1[i-1] and nums2[j] because when i == 0, nums1[i-1] does not exist, and when
 * j == n, nums2[j] does not exist.
 * if i == m || j == 0, no need to compare nums1[i] and nums2[j-1] for similar reason
 *
 * Rewrite the goal:
 * Find i such that
 * (i == 0 || j == n || nums1[i-1] <= nums2[j]) && (i == m || j == 0 || nums1[i] >= nums2[j-1]), where j=(m+n+1)/2-i
 *
 * Algorithm using Binary search idea:
 * initialization: l=0, r=m-1
 * while l<=r
 *   calculate i=(l+r)/2, j=(m+n+1)/2-i;
 *
 *   (1) if(i>0 && j<n && nums1[i-1]>nums2[j]) decrease i by moving r to i;
 *   (2) if(i<m && j>0 && nums1[i]<nums2[j-1]) increase i by moving l to i;
 *   (3) remaining case: (i==0 || j==n || nums1[i-1]<=nums2[j]) && (i==m || j==0 || nums1[i]>=nums2[j-1])
 * end loop
 */

```

```

public double findMedianSortedArrays(int[] nums1, int[] nums2) {
    if(nums1 == null || nums2 == null || (nums1.length == 0 && nums2.length == 0)) return 0;

    int m=nums1.length, n=nums2.length;
    if(m > n){
        return findMedianSortedArrays(nums2, nums1);
    }

    if(m == 0){
        if(n%2==0){
            return (nums2[(n-1)/2] + nums2[n/2])/2.0;
        }else{
            return nums2[n/2];
        }
    }

    int l = 0, r = m; //we allow i=0 and i=m
    int i=0, j=0;
    while(l <= r){
        i = l+(r-l)/2;
        j = (m+n+1)/2 - i;
        if(i>0 && j<n && nums1[i-1]>nums2[j]){
            r=i;
        }else if(i<m && j>0 && nums1[i]<nums2[j-1]){
            l=i+1;
        }else{
            break;
        }
    }


    int left_max;

    if(i==0){
        left_max=nums2[j-1];
    }else if(j==0){
        left_max=nums1[i-1];
    }else{
        left_max=nums1[i-1] > nums2[j-1]?nums1[i-1]:nums2[j-1];
    }
    if((m+n)%2!=0){
        return left_max;
    }

    int right_min;
    if(i==m){
        right_min=nums2[j];
    }else if(j==n){
        right_min=nums1[i];
    }else{
        right_min=nums1[i]<nums2[j]?nums1[i]:nums2[j];
    }

    return (left_max+right_min)/2.0;
}

```

 每日一题算法题

[最佳提交](#)


```

/**
 *
 * Author:@Jiashen
 * Time/Space: O(log(m+n))/O(log(m+n))
 * Method:Binary Search, Divide and Conquer
 *
 * 1. nums1[k/2-1] == nums2[k/2-1], return nums1[k/2-1] or nums2[k/2-1]
 * 2. nums1[k/2-1] < nums2[k/2-1], means nums1[0...k/2-1] at the front of k, so delete nums1[0...k/2-1].
 * 3. nums1[k/2-1] > nums2[k/2-1], means nums2[0...k/2-1] at the front of k, so delete nums2[0...k/2-1].
 *
 * Loop termination conditions:
 * 1. when nums1 or nums2 equals null, return directly nums1[k-1] or nums2[k-1].
 * 2. when k == 1, return min(nums1[0], nums2[0]).
 * 3. when nums1[k/2-1] == nums2[k/2-1], return nums1[k/2-1] or nums2[k/2-1].
 */
public class Solution {
    public double findMedianSortedArrays(int[] nums1, int[] nums2) {
        int m = nums1.length;
        int n = nums2.length;
        int total = m+n;
        if((total & 0x1) == 1){
            return find_kth(nums1, m, nums2, n, total/2 + 1);
        } else {
            return (find_kth(nums1, m, nums2, n, total/2)+
                    find_kth(nums1, m, nums2, n, total/2 + 1))/2.0;
        }
    }
    private int find_kth(int[] nums1, int m, int[] nums2, int n, int k){
        // loop termination conditions
        if(m > n) return find_kth(nums2, n, nums1, m, k);
        if(m == 0) return nums2[k-1];
        if(k == 1) return Math.min(nums1[0], nums2[0]);

        //divide k into two parts

        int a = Math.min(k/2, m), b = k - a;
        if(nums1[a-1] < nums2[b-1])
            return find_kth(Arrays.copyOfRange(nums1,a,m), m-a, nums2, n, k-a);
        else if(nums1[a-1] > nums2[b-1])
            return find_kth(nums1, m, Arrays.copyOfRange(nums2,b,n), n-b, k-b);
        else
            return nums1[a-1];
    }
}

```



活动预告

二群的 @黄xing 会于2017年5月4号北京时间晚上9点以直播的形式分析和讲解这道题。在直播开始之前，我会将直播地址同步到的每日一题算法群中，敬请期待。