Median of Two Sorted Arrays

by bugnofree

Publish → 2019-02-23 Update → 2019-02-23

1. 题目描述

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)).

You may assume nums1 and nums2 cannot be both empty.

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

2. 题解

题目里面说给的是两个排好序的数组 M, N, 但是对于这两个数组都是升序, 都是降序,还是一升一降则没有说明, 但是根据讨论区的情况, 大家默认是两个数组要么是同升序,要么同降序处理的.

2.1. topk 求解

既然是两个有序数组, 那么只需要合并, 然后取数即可.

但是合并操作的时间复杂度为 O(m + n), 题目要求的是 O(log(m + n)).

这个可以用 topk 的思想求解, 参照 http://ahageek.com/writer/migrat....

这个思路是我两年前做的.

2.2. 中位数定义法

按照 https://leetcode.com/problems/me...这里说的 O(log(min(m, n))) 的算法, 只适用于两个数组都是升序的情况,leetcode 上面虽然没说都是升序的, 但是似乎就是按照升序来判定的.

这种方法太强了, 但边界条件比较不好处理.

主要思想如下, 给定中位数, 那么在该中位数两侧的数字个数是一样多的.

对于两个升序数组 S, L (假设 S 是较长的一个, L 是较短的一个), 长度分别为 m, n.

那么在 S 的索引 i 处切割, 在 L 的索引 j 处切割, 将会得到四个数组

```
S[0..i-1], S[i..n-1]
L[0..j-1], L[j..m-1]
```

那这样的话, 只要我们能保证处于低侧的数组(S[0..i-1], L[0..j-1])的最大值小于高侧数组,且低侧数组元素数等于高侧元素数, 那么我们就找到了最佳的分割值 i.

有一个问题是, 假如 m+n 是奇数该怎么办? 无论怎么划分, 低侧和高侧的数目都不可能相等,因为不可能对一个数分割, 对于奇数的情况, 划分后必有一侧多一个数, 那么是那一侧呢?

我们用一个简单的例子看一下

```
1 3 | 9
7 8 | 11 13
```

这个小例子, 最佳 i = 2(此时可求出 j = 2), 低侧比高侧多 1.

也就是说, 如果 m+n 为奇数, 我们只需要返回低侧的最大值即可.

但尽管如此, 我们求 j 的时候, 不用考虑 m+n 是奇数还是偶数,因为 (m+n)/2 的值和 (m+n+1)/2 的值是一样的.

如果是偶数的话, 我们需要计算低侧最大值和高侧最小值, 然后求其平均数返回.

这道题的优化在于寻找 i 值, 用了二分法寻找, 所以能够把时间优化到 log 级别.

最终 AC 代码如下: 时间复杂度 `O(log(min(m, n)))` .

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

class Solution {
public:
```

```cpp
// two ascending arrays, they cannot be both empty
double findMedianSortedArrays(vector<int>& nums1, vector<int>& nums2) {
    if(nums1.size() == 0 || nums2.size() == 0) {
        vector<int> &validarr = nums1.size() == 0 ? nums2 : nums1;
        if(validarr.size() % 2 == 0) {
            return (validarr[validarr.size() / 2] + validarr[validarr.size() / 2 - 1]) / 2.0;
        } else {
            return validarr[validarr.size() / 2.0];
        }
    }

    vector<int> &shorter = nums1.size() > nums2.size() ? nums2 : nums1;
    vector<int> &longer = nums1.size() > nums2.size() ? nums1 : nums2;

    int l = 0, r = shorter.size();
    int i, j;
    while(l <= r) {
        i = (l + r) / 2;
        j = (shorter.size() + longer.size() + 1) / 2 - i;

        // decrease i
        if(i > 0 && j < longer.size() && shorter[i - 1] > longer[j]) {
            r = i - 1;
        // increase i
        } else if(j > 0 && i < shorter.size() && longer[j - 1] > shorter[i]) {
            l = i + 1;
        // catch that(with egde cases)
        } else {
            int maxl;
            if(i == 0) {
```

```
                    maxl = longer[j - 1];
                } else if(j == 0) {
                    maxl = shorter[i - 1];
                } else {
                    maxl = max(shorter[i - 1], longer[j - 1]);
                }

                // in this case, number_of_left_part == number_of_right_part + 1
                // take the following as an example to understand that:
                // 1 3 | 9     <= i = 2
                // 7 8 | 11 13 <= j = 2
                if((shorter.size() + longer.size()) % 2 == 1) {
                    return maxl;
                }

                int minr;
                if(i == shorter.size()) {
                    minr = longer[j];
                } else if(j == longer.size()) {
                    minr = shorter[i];
                } else {
                    minr = min(shorter[i], longer[j]);
                }
                return (maxl + minr) / 2.0;
            }
        }
        return 0.0;
    }
};
```

```cpp
int main() {
    Solution sol;
    vector<int> nums1 = {1, 3, 9};
    vector<int> nums2 = {7, 8, 11, 13};
    double s = sol.findMedianSortedArrays(nums1, nums2);
    cout << s << endl;
    return 0;
}
```