Previous Next

350. Intersection of Two Arrays II

Oct. 21, 2019 | 47.1K views

Average Rating: 4.75 (64 votes)

Given two arrays, write a function to compute their intersection.

Example 1:

```
Input: nums1 = [1,2,2,1], nums2 = [2,2]
Output: [2,2]
```

Example 2:

```
Input: nums1 = [4,9,5], nums2 = [9,4,9,8,4]
Output: [4,9]
```

Note:

- Each element in the result should appear as many times as it shows in both arrays.
- The result can be in any order.

Follow up:

- What if the given array is already sorted? How would you optimize your algorithm?
- What if *nums1*'s size is small compared to *nums2*'s size? Which algorithm is better?
- What if elements of *nums2* are stored on disk, and the memory is limited such that you cannot load all elements into the memory at once?

Solution

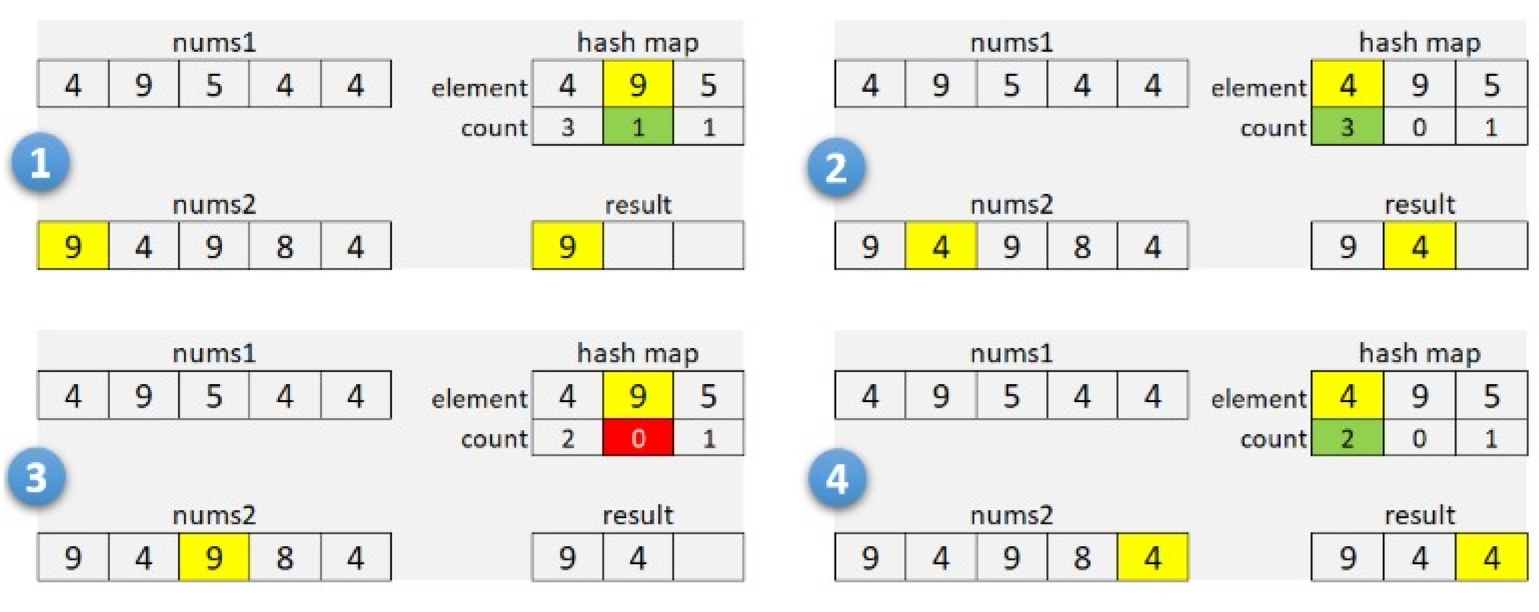
If an interviewer gives you this problem, your first question should be - *how should I handle duplicates?* Your second question, perhaps, can be about the order of inputs and outputs. Such questions manifest your problem-solving skills, and help you steer to the right solution.

The solution for the previous problem, 349. Intersection of Two Arrays, talks about approaches when each number in the output must be unique. For this problem, we need to adapt those approaches so that numbers in the result appear as many times as they do in both arrays.

Approach 1: Hash Map

For the previous problem, we used a hash set to achieve a linear time complexity. Here, we need to use a hash map to track the count for each number.

We collect numbers and their counts from one of the arrays into a hash map. Then, we iterate along the second array, and check if the number exists in the hash map and its count is positive. If so - add the number to the result and decrease its count in the hash map.



> It's a good idea to check array sizes and use a hash map for the smaller array. It will reduce memory usage when one of the arrays is very large.

Algorithm

1. If `nums1` is larger than `nums2`, swap the arrays.
2. For each element in `nums1`:
   - Add it to the hash map `m`.
     - Increment the count if the element is already there.
3. Initialize the insertion pointer (`k`) with zero.
4. Iterate along `nums2`:
   - If the current number is in the hash map and count is positive:
     - Copy the number into `nums1[k]`, and increment `k`.
     - Decrement the count in the hash map.
5. Return first `k` elements of `nums1`.

> For our solutions here, we use one of the arrays to store the result. As we find common numbers, we copy them to the first array starting from the beginning. This idea is from this solution by sankitgupta.

C++ Java Copy

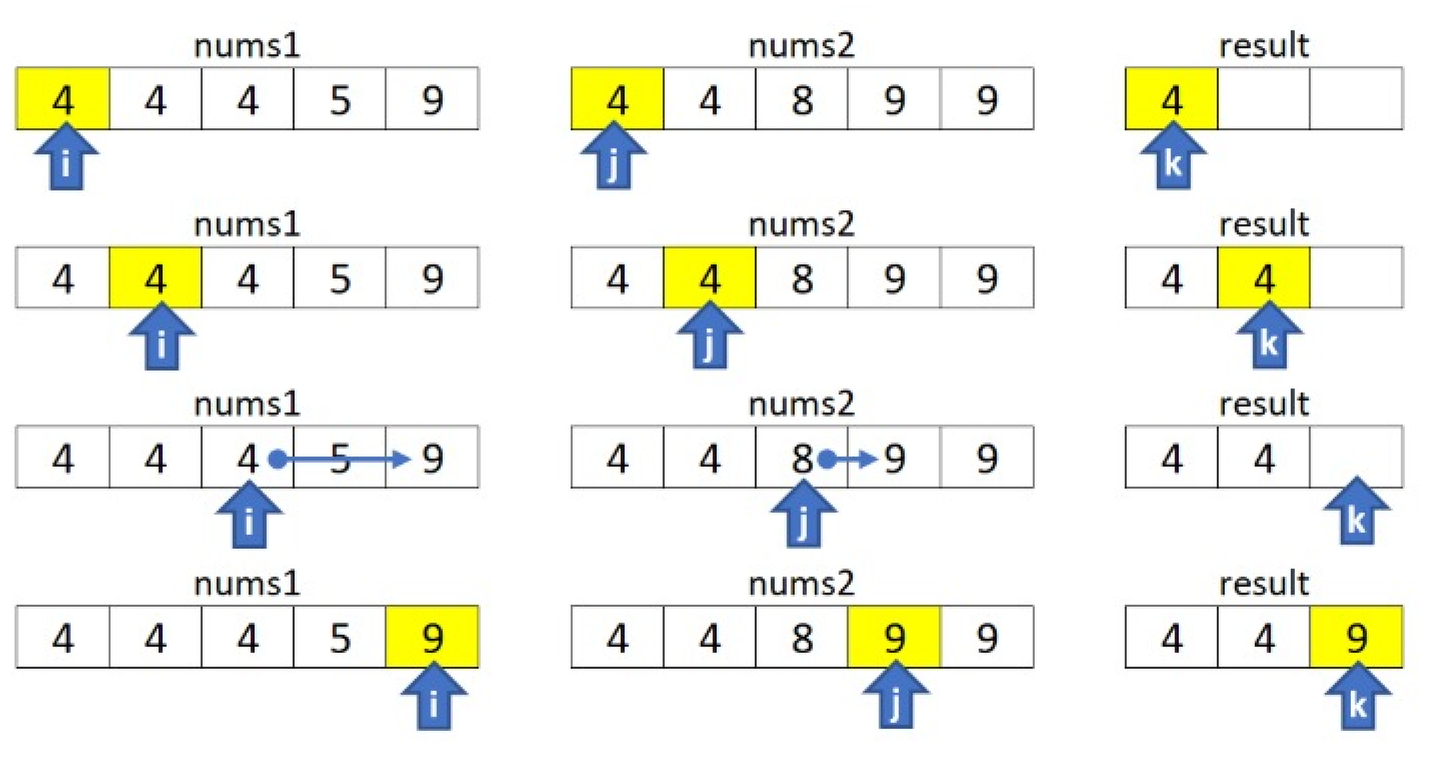
```
vector<int> intersect(vector<int>& nums1, vector<int>& nums2) {
    if (nums1.size() > nums2.size()) {
        return intersect(nums2, nums1);
    }
    unordered_map<int, int> m;
    for (auto n : nums1) {
        ++m[n];
    }
    int k = 0;
    for (auto n : nums2) {
        auto it = m.find(n);
        if (it != end(m) && --it->second >= 0) {
            nums1[k++] = n;
        }
    }
    return vector(begin(nums1), begin(nums1) + k);
}
```

Complexity Analysis

- Time complexity: $\mathcal{O}(n + m)$, where n and m are the lengths of the arrays. We iterate through the first, and then through the second array; insert and lookup operations in the hash map take a constant time.
- Space complexity: $\mathcal{O}(\min(n, m))$. We use hash map to store numbers (and their counts) from the smaller array.

Approach 2: Sort

You can recommend this method when the input is sorted, or when the output needs to be sorted. Here, we sort both arrays (assuming they are not sorted) and use two pointers to find common numbers in a single scan.



Algorithm

1. Sort `nums1` and `nums2`.
2. Initialize `i`, `j` and `k` with zero.
3. Move indices `i` along `nums1`, and `j` through `nums2`:
   - Increment `i` if `nums1[i]` is smaller.
   - Increment `j` if `nums2[j]` is smaller.
   - If numbers are the same, copy the number into `nums1[k]`, and increment `i`, `j` and `k`.
4. Return first `k` elements of `nums1`.

C++ Java Copy

```
vector<int> intersect(vector<int>& nums1, vector<int>& nums2) {
    sort(begin(nums1), end(nums1));
    sort(begin(nums2), end(nums2));
    int i = 0, j = 0, k = 0;
    while (i < nums1.size() && j < nums2.size()) {
        if (nums1[i] < nums2[j]) {
            ++i;
        } else if (nums1[i] > nums2[j]) {
            ++j;
        } else {
            nums1[k++] = nums1[i++];
            ++j;
        }
    }
    return vector<int>(begin(nums1), begin(nums1) + k);
}
```

Complexity Analysis

- Time complexity: $\mathcal{O}(n\log{n} + m\log{m})$, where n and m are the lengths of the arrays. We sort two arrays independently, and then do a linear scan.
- Space complexity: $\mathcal{O}(1)$. We sort the arrays in-place. We ignore the space to store the output as it is not essential to the algorithm itself.

Approach 3: Built-in Intersection

This is similar to Approach 2. Instead of iterating with two pointers, we use a built-in function to find common elements. In C++, we can use `set_intersection` for sorted arrays (or multisets).

The retainAll method in Java, unfortunately, does not care how many times an element occurs in the other collection. You can use the retainOccurrences method of the multiset implementation in Guava.

Algorithm

> Note that `set_intersection` returns the position past the end of the produced range, so it can be used as an input for the `erase` function. The idea is from this solution by StefanPochmann.

C++ Copy

```
vector<int> intersect(vector<int>& nums1, vector<int>& nums2) {
    sort(begin(nums1), end(nums1));
    sort(begin(nums2), end(nums2));
    nums1.erase(set_intersection(begin(nums1), end(nums1),
        begin(nums2), end(nums2), begin(nums1)), end(nums1));
    return nums1;
}
```

Complexity Analysis

- Same as for approach 2 above.

Follow-up Questions

1. What if the given array is already sorted? How would you optimize your algorithm?
   - We can use either Approach 2 or Approach 3, dropping the sort of course. It will give us linear time and constant memory complexity.
2. What if *nums1's* size is small compared to *nums2's* size? Which algorithm is better?
   - Approach 1 is a good choice here as we use a hash map for the smaller array.
3. What if elements of *nums2* are stored on disk, and the memory is limited such that you cannot load all elements into the memory at once?
   - If `nums1` fits into the memory, we can use Approach 1 to collect counts for `nums1` into a hash map. Then, we can sequentially load and process `nums2`.
   - If neither of the arrays fit into the memory, we can apply some partial processing strategies:
     - Split the numeric range into subranges that fits into the memory. Modify Approach 1 to collect counts only within a given subrange, and call the method multiple times (for each subrange).
     - Use an external sort for both arrays. Modify Approach 2 to load and process arrays sequentially.

Rate this article:

Previous Next

Comments: 13 Sort By

Type comment here... (Markdown is supported)

Preview Post

bobxu1128 187 October 23, 2019 2:47 AM

good article.

23 Share Reply

hhbagels 121 December 23, 2019 1:57 AM Report

So this problem falls under easy category? There are lot of hards and mediums easier than this problem!

25 Share Reply

SHOW 1 REPLY

mchen2 10 October 24, 2019 9:20 AM

so why we tagged it as binary search??

10 Share Reply

SHOW 2 REPLIES

ufdeveloper 59 March 2, 2020 4:54 AM

The question is so incomplete! The intersection is not defined correctly at all! How is it an intersection if there are other elements in between the intersecting elements?

8 Share Reply

SHOW 2 REPLIES

notandor 4 March 12, 2020 12:10 AM

Thanks for various solutions writeup.
But it is very bad design to change passed in input array content. Immediate red flag.
Create a new array with length of minimum of 2 arrays:

Read More

3 Share Reply

kevinYang19 2 May 14, 2020 6:28 AM

Nice article and good solutions for the follow up questions.
thanks a lot

2 Share Reply

rbethamcharla 2 April 8, 2020 7:49 AM

In the first solution with Hashmap, We decided it based on count. But a subarray should follow the order also. How we address this?

2 Share Reply

SHOW 1 REPLY

pavan_kumar 1 May 18, 2020 10:43 PM

If number1.size() is larger than number2.size().
Why do we want to perform a swap?
because even if nums1 size is greater what if it contains everything as a duplicate.

nums1 = [1,1,1,1,1,1,1,1,1]

Read More

1 Share Reply

SHOW 1 REPLY

jskiertsky 3 November 11, 2019 10:32 PM

In Javascript, I wrote a reduce function, then wrote a custom callback that I passed to reduce to the specifications of the question.

I'm sure there are lots of ways of doing this. It's kind of like implementing the written solution of using a premade intersection in another language, but the advantage of rolling your own is that (a) it looks

Read More

1 Share Reply

supereagle 8 November 9, 2019 11:50 PM

I can't understand the last follow up Question's answer, when both arrays can't fit in the memory.
Can anybody explain please?

1 Share Reply

SHOW 5 REPLIES

‹ 1 2 ›