Duplicate Elements of An Array

One common misunderstanding is that coding interview is all about solving algorithm questions. In fact, the answer itself is only part of the evaluation and sometimes it is not the most important part at all.

There are many other factors being evaluated during an interview. For instance, your analysis process is at least equally important. More specifically, interviewers care a lot about how you approach a problem step by step, how you optimize your solution, how you compare different approaches and so on so forth.

So in this post, we want to focus more on discussion and analysis. You will learn a lot about what I mean by "solution is not important". We start with a simple question, but there are a bunch of follow-up questions after that.

Question

Given an array of string, find duplicate elements.

For instance, in array ["abc", "dd", "cc", "abc", "123"], the duplicate element is "abc". Let's start with this simple scenario and I'll cover more follow-up questions soon. Also, as before, we only select questions that are asked by top companies. This one was asked by Uber, Facebook, Amazon recently.

Solution

I'll skip the O(N^2) brute force solution that you compare each of two strings because it's too obvious. One common technique is the **trade-off between time and space**. Since we want to make the algorithm faster, we can think of how to use more memory to achieve this.
I hope when you see "find duplicate", you can think of hash set immediately since hash is the most common technique to detect duplicates. If we store every element into a hash set, we can make it O(N) for both time and space complexity.

File

Let's extend this question a little bit. **What if the array is too large to put in memory?** Apparently, we have to store all those strings in files. Then how can we find duplicate elements?

Many people have almost no experience with "big data" that cannot fit into memory. But no worries, you will see the problem is not as hard as you thought. Let's think about it in this way. We can load as many data as possible into memory and find duplicates with the same approach above, however, the problem is that we can't compare data from separate batches. Does this problem sound familiar to you?

Again, I hope you can think about external merge sort (https://en.wikipedia.org/wiki/External_sorting), which solves exactly the same problem. Ok, the most obvious solution is to do an external sort over all the strings and then we can just compare adjacent strings to find duplicates.

File pivot

There's another way to do that. Since we can only load limited data into memory, we can only load strings that are possible to be duplicate. Let's say we can pick k pivots like quick sort (https://en.wikipedia.org/wiki/Quicksort). Each time, we only load strings that are between [pivot i, pivot i+1] into memory and find duplicates if any.

How do we select k? We need to make sure each bucket can fit into memory, otherwise, we need to divide the bucket into multiple ones.

How do we evaluate the efficiency? Unlike normal big-O analysis, when file operation is involved, the bottleneck is always how many times of file operations are used. So there's no obvious answer which approach is better, as long as you are trying to estimate the number of file operations, it's good.

Distributed system

Let's go one step further. **What if the array is too large to store on one machine and we have to distribute it to multiple nodes?** You will see how similar the problem is to the in-disk version.

We can first sort arrays in each of the machines. Then, we select a master machine and all the other machines send each string element one by one to the master in order. Thus, the master machine can easily find duplicate elements. This is exactly the same as the external merge sorting except it is using network to communicate.

Similarly, we can also split the array into shards and each machine stores one shard. More specifically, suppose machine k stores strings from "1000" to "5000", then every other machine is responsible for sending strings within this range to machine k via network. Once it's done, we can just find duplicate strings within a single machine. This is same as the pivot solution.

Evaluation

How do you evaluate the performance of the algorithm? This is not an easy question since in distributed systems there are quite a few factors we need to consider. The basic idea is that we need to quickly pinpoint the bottleneck. In a single machine, the key is to reduce the number of file operations. In a distributed system, more often than not the key is to reduce network requests.

If you can try to estimate the number of network requests needed with some reasonable assumption, interviewers will be impressed for sure. As you can see, for many interview questions, there's no clear answer and even interviewers don't know the solution. The point here is that as long as you are trying to solve the problem and provide reasonable analysis, you will get a good score.

Takeaways

I think the most important takeaway is to know that analysis is way important than the solution. As an interviewer, I don't really like to hear answers like "I don't know". Instead, I'd like to see that candidates try hard to figure out the solution and keep telling me what's in hid mind.

Besides, all the techniques used here like external merge sort are very common for disk problems and distributed system problems. You should not be scared when asking what if we scale this problem to disk or multiple machines.

Another advice is that whenever you solve some questions, try to ask yourself what if we expand the question to a larger scale.

Related Posts:

1. **Group Anagrams (http://blog.gainlo.co/index.php/2016/05/06/group-anagrams/)**
2. **3Sum (http://blog.gainlo.co/index.php/2016/07/19/3sum/)**
3. **Minimum Number of Deletions Of a String (http://blog.gainlo.co/index.php/2016/04/29/minimum-number-of-deletions-of-a-string/)**
4. **Flatten a Linked List (http://blog.gainlo.co/index.php/2016/06/12/flatten-a-linked-list/)**

Name *

Email *

Website

Post Comment

Group Anagrams (http://blog.gainlo.co/index.php/2016/05/06/group-anagrams/)

Design a Cache System (http://blog.gainlo.co/index.php/2016/05/17/design-a-cache-system/)