Minimum Number of Deletions Of a String

Another string question in our coding interview questions collection. It seems that string is getting really popular and many companies like Google, Facebook are asking about it in recent interviews.

After a second thought, this makes sense in fact. String is a quite flexible data structure and many concepts can be covered from a string problem like hash, memory and so on so forth. In addition, it's also a data structure you're gonna use almost every day. That's why many string interview questions are quite relevant to real world projects.

In this post, we're going to talk about topics including string manipulation, dictionary, time complexity etc. and in the end, I'll summarize several commonly used techniques as before.

Question

Given a dictionary and a word, find the minimum number of deletions needed on the word in order to make it a valid word.

For example, string "catn" needs one deletion to make it a valid word "cat" in the dictionary. And string "bcatn" needs two deletions.

Dictionary has always been an interesting topic in string interview problems, which is part of the reason I'd like to cover this here. Also, this question was asked by Google recently.

Dictionary

Given that dictionary is so common in coding interview questions that I'd like to briefly summarize few strategies/techniques here.

- To store a dictionary, usually people will use data structures including Hash set, Trie or maybe just array. You'd better understand pros and cons of each of them.
- You may choose to have a pre-processing step to read the whole dictionary and store into your preferred data structure. Since once it's loaded, you can use it as many times as you want.
- If the dictionary is not too large, you may take the dictionary traverse time as a constant.

Traverse dictionary

Coming back to this problem, if we assume the dictionary can be traversed quickly (not too many entries), one approach is to go through each word in the dictionary, calculate the number deletions required, and return the minimum one.

To calculate the number of deletions efficiently, we'll use the common technique here. One fact is that if a longer string can be transformed to a shorter one by deleting characters, the longer string must contain all the characters of the smaller one in order. If you have noticed this fact, then you should know that we only need to traverse the two strings once in order to get the deletion number.

More specifically, we put two indices (L for the longer string, S for the shorter string) pointing to beginning of each string. If the two characters under the indices are different, move L forward by one character. If the two characters are same, move both forward. If S comes to the end, it means the longer string contains all the characters in order, so the number of deletion needed is just len(longer) – len(shorter).

Assuming the size of the dictionary is M and length of the given word is N, the time complexity is O(MN) because for each word in the dictionary, we may need to iterate over the given word.

Traverse the word

What if the dictionary is really large? Actually we can solve this problem from the other side – traverse all the possible words generated from deletion of the given word.

So for the given word, we try to delete each of the characters and check if the new word exists in the dictionary. **Since we need to quickly check the existence of a work in dictionary, we need to load the dictionary into a hash set.**

So the time complexity for pre-processing is O(M) (traverse the whole dictionary once) and for the rest of the algorithm is O(2^N) because we need to get all the possible words generated from the given word. It's also worth to note that once the dictionary is loaded, we don't need to do the pre-processing again and that's why sometimes we can ignore the time spent here.

So which solution is better? It depends on the size of the dictionary and length of the given word.

Takeaways

To sum up some techniques in this question:

- You should be aware of common data structures for dictionary and pros and cons of each of them.

- Given that the size of the dictionary is fixed, it’s not a bad idea to just iterate over it.
- Having two indices to traverse/compare two string/arrays is quite common. For example, we use the same approach to merging two sorted arrays.

You may notice that it’s not easy to write the code for “traverse the word” solution. So please try to finish the code for this part.

The post Minimum Number of Deletions Of a String appeared first on Gainlo Mock Interview Blog.

Group Anagrams

This is another post in the coding interview questions collection. In this series, we’ll cover recent hot questions from top companies like Google, Facebook, Über, Linkedin etc.. More importantly, the goal of these posts is not giving you something like a standard answer.

Instead, we focus on telling you how to analyze each question and how to re-use the same techniques in similar problems. At the end of each post, we’ll summarize some common strategies used in the question.

In this post, we are going to cover topics including hash map, string manipulation and sorting as well.

Question

Given a set of random string, write a function that returns a set that groups all the anagrams *together.*

For example, suppose that we have the following strings:
“cat”, “dog”, “act”, “door”, “odor”

Then we should return these sets: {“cat”, “act”}, {“dog”}, {“door”, “odor”}.

Few reasons I selected this problem:

- It was asked by Facebook a month ago.
- Anagram is a really popular topic in recent interviews.
- Tons of techniques used in this problem can be reused in similar questions.

Again, try to think about this problem before moving on.

Anagram

If you keep following our blog posts, this shouldn’t be the first time you see anagrams. In question If a String Contains an Anagram of Another String, we also covered this topic and some techniques will be used here as well.

If you have tried with some examples in this question, you should notice that the key is to check if two strings are anagram because with this issue solved, you can easily tell which strings should be grouped together.

To check if two strings are anagram – with same set of characters, one approach is to sort all characters and then compare if two sorted strings are identical. Since you will need to output the original string, you may need to keep it together with the sorted string. Therefore, we have this initial idea:

1. Transform each string to a tuple (sorted string, original string). For instance, “cat” will be mapped to (“act”, “cat”).
2. Sort all the tuples by the sorted string, thus, anagrams are grouped together.
3. Output original strings if they share the same sorted string.

Optimization

In fact, you will notice that step 2 is not efficient – O(nlogn) time complexity for sorting. In order to make it in linear time, you can use a hash map whose key is the sorted string and value is an array of corresponding original strings.

By doing this, you can reduce the time complexity of step 2 to O(N). However, the downside is that you need more space to store the hash map. Given that in step 1 we already need extra space (O(N)) for the sorted string, a hash map won’t change the final space complexity.

From our previous post, we mentioned about another simple approach to check anagram. If we map each character to a prime number and the whole string is mapped to the multiples of all the prime numbers of its characters, anagrams should have the same multiple. The benefit of this approach is that we can check if two strings are anagrams in linear time instead of O(MlogM) by sorting (M is the length of a string).

Time space trade-off

This question is a perfect example of time space trade-off. With a hash map, we can reduce the time complexity to linear, which is true for both the overall grouping and anagram checking. However, it requires additional space. Without a hash map, we need to do the sorting, which is slower.

The idea here is that when we want to make the algorithm faster, one direction to think about is to use additional space. Hash map or hash set are one of the most common data structures to consider. On the flip side, if we want to reduce usage of memory, we may consider slower the program.

Takeaways

As before, let’s summarize few techniques we used in this question:

- When we need to group similar things together, I expect data structures like hash map to come to your mind in 1 second. This is commonly used not only in coding interview questions but real life projects as well.

- To check anagrams, we can use the prime number approach or sorting.
- Time space trade-off is a very common approach when optimizing algorithms.

The post Group Anagrams appeared first on Gainlo Mock Interview Blog.

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 Über, 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, 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. 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.

The post Duplicate Elements of An Array appeared first on Gainlo Mock Interview Blog.