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, Uber, 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 (https://en.wikipedia.org/wiki/Anagram) 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 (http://blog.gainlo.co/index.php/2016/04/08/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 (http://blog.gainlo.co/index.php/2016/04/08/if-a-string-contains-an-anagram-of-another-string/), 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.

Related Posts:

1. **Find The Longest Substring With K Unique Characters (http://blog.gainlo.co/index.php/2016/04/12/find-the-longest-substring-with-k-unique-characters/)**
2. **If a String Contains an Anagram of Another String (http://blog.gainlo.co/index.php/2016/04/08/if-a-string-contains-an-anagram-of-another-string/)**
3. **Duplicate Elements of An Array (http://blog.gainlo.co/index.php/2016/05/10/duplicate-elements-of-an-array/)**
4. **Minimum Number of Deletions Of a String (http://blog.gainlo.co/index.php/2016/04/29/minimum-number-of-deletions-of-a-string/)**

---

2 thoughts on "Group Anagrams"

CEM
May 7, 2016 at 3:38 am (http://blog.gainlo.co/index.php/2016/05/06/group-anagrams/#comment-784)

It might also be a good idea to mention that using the prime number approach may cause overflow problems very quickly if the strings are long enough or the numbers associated with the characters in the string are large enough. If we say that a corresponds to 2, the first prime number, and z corresponds to the 26th prime number (101), then even a string like 'zzz' will easily overflow an unsigned integer in a 32-bit machine.

Reply (http://blog.gainlo.co/index.php/2016/05/06/group-anagrams/?replytocom=784#respond)

Reply (http://blog.gainlo.co/index.php/2016/05/06/group-anagrams/?replytocom=784#respond)

CEM
May 7, 2016 at 3:39 am (http://blog.gainlo.co/index.php/2016/05/06/group-anagrams/#comment-785)

Sorry, but I meant 'zzzzz'. 3 'z's will not cause an overflow.

Reply (http://blog.gainlo.co/index.php/2016/05/06/group-anagrams/?replytocom=785#respond)

LEAVE A REPLY

Your email address will not be published. Required fields are marked *

Comment

Name *

Email *

Website

Post Comment

How to Design a Trending Algorithm for Twitter (http://blog.gainlo.co/index.php/2016/05/03/how-to-design-a-trending-algorithm-for-twitter/)

Duplicate Elements of An Array (http://blog.gainlo.co/index.php/2016/05/10/duplicate-elements-of-an-array/)