242. Valid Anagram

March 5, 2016 | 225.8K views

Previous Next

Given two strings *s* and *t* , write a function to determine if *t* is an anagram of *s*.

Example 1:

```
Input: s = "anagram", t = "nagaram"
Output: true
```

Example 2:

```
Input: s = "rat", t = "car"
Output: false
```

Note:
You may assume the string contains only lowercase alphabets.

Follow up:
What if the inputs contain unicode characters? How would you adapt your solution to such case?

Solution

Approach #1 (Sorting) [Accepted]

Algorithm

An anagram is produced by rearranging the letters of *s* into *t*. Therefore, if *t* is an anagram of *s*, sorting both strings will result in two identical strings. Furthermore, if *s* and *t* have different lengths, *t* must not be an anagram of *s* and we can return early.

```
public boolean isAnagram(String s, String t) {
    if (s.length() != t.length()) {
        return false;
    }
    char[] str1 = s.toCharArray();
    char[] str2 = t.toCharArray();
    Arrays.sort(str1);
    Arrays.sort(str2);
    return Arrays.equals(str1, str2);
}
```

Complexity analysis

- Time complexity : $O(n \log n)$. Assume that n is the length of *s*, sorting costs $O(n \log n)$ and comparing two strings costs $O(n)$. Sorting time dominates and the overall time complexity is $O(n \log n)$.
- Space complexity : $O(1)$. Space depends on the sorting implementation which, usually, costs $O(1)$ auxiliary space if `heapsort` is used. Note that in Java, `toCharArray()` makes a copy of the string so it costs $O(n)$ extra space, but we ignore this for complexity analysis because:
    - It is a language dependent detail.
    - It depends on how the function is designed. For example, the function parameter types can be changed to `char[]`.

Approach #2 (Hash Table) [Accepted]

Algorithm

To examine if *t* is a rearrangement of *s*, we can count occurrences of each letter in the two strings and compare them. Since both *s* and *t* contain only letters from *a* — *z*, a simple counter table of size 26 is suffice.

Do we need *two* counter tables for comparison? Actually no, because we could increment the counter for each letter in *s* and decrement the counter for each letter in *t*, then check if the counter reaches back to zero.

```
public boolean isAnagram(String s, String t) {
    if (s.length() != t.length()) {
        return false;
    }
    int[] counter = new int[26];
    for (int i = 0; i < s.length(); i++) {
        counter[s.charAt(i) - 'a']++;
        counter[t.charAt(i) - 'a']--;
    }
    for (int count : counter) {
        if (count != 0) {
            return false;
        }
    }
    return true;
}
```

Or we could first increment the counter for *s*, then decrement the counter for *t*. If at any point the counter drops below zero, we know that *t* contains an extra letter not in *s* and return false immediately.

```
public boolean isAnagram(String s, String t) {
    if (s.length() != t.length()) {
        return false;
    }
    int[] table = new int[26];
    for (int i = 0; i < s.length(); i++) {
        table[s.charAt(i) - 'a']++;
    }
    for (int i = 0; i < t.length(); i++) {
        table[t.charAt(i) - 'a']--;
        if (table[t.charAt(i) - 'a'] < 0) {
            return false;
        }
    }
    return true;
}
```

Complexity analysis

- Time complexity : $O(n)$. Time complexity is $O(n)$ because accessing the counter table is a constant time operation.
- Space complexity : $O(1)$. Although we do use extra space, the space complexity is $O(1)$ because the table's size stays constant no matter how large *n* is.

Follow up

What if the inputs contain unicode characters? How would you adapt your solution to such case?

Answer

Use a hash table instead of a fixed size counter. Imagine allocating a large size array to fit the entire range of unicode characters, which could go up to more than 1 million. A hash table is a more generic solution and could adapt to any range of characters.

Rate this article:

Previous Next

Comments: 88 Sort By

Type comment here... (Markdown is supported)

Preview Post

dengjiangzhouwai 92 November 30, 2018 11:53 AM

My Python solution:

```
def isAnagram(self, s, t):
    """
    :type s: str
    :type t: str
    :rtype: bool
    """
```

Read More

48 Share Reply

SHOW 7 REPLIES

terriblewhiteboard 3841 May 3, 2020 5:51 PM

I made a video if anyone is having trouble understanding the solution
https://www.youtube.com/watch?v=FMNuwJAQ2pE&feature=youtu.be

Read More

26 Share Reply

viccco 33 March 25, 2019 11:27 AM

Hash method is really a intelligent one!

36 Share Reply

SHOW 2 REPLIES

beethovenzhang 84 October 26, 2018 7:32 AM

Solution for the follow up question

```
class Solution {
    public boolean isAnagram(String s, String t) {
        int[] table = new int[26];
        for (int i = 0; i < s.length(); i++) {
            table[s.charAt(i) - 'a']++;
        }
        for (int i = 0; i < t.length(); i++) {
            table[t.charAt(i) - 'a']--;
            if (table[t.charAt(i) - 'a'] < 0) {
                return false;
            }
        }
        return true;
    }
}
```

Read More

21 Share Reply

SHOW 7 REPLIES

wuwei41 7 July 1, 2019 10:37 AM

One liner:
return collections.Counter(s) == collections.Counter(t)

7 Share Reply

ajinkya93 45 July 31, 2018 8:41 AM

Python3

class Solution:
def isAnagram(self, string1, string2):

Read More

5 Share Reply

SHOW 1 REPLY

Stick_to_it 1 March 12, 2019 9:21 PM

However, I don't understand why the Space complexity of Approach 1 is O(1). O<O

4 Share Reply

SHOW 6 REPLIES

AbelValdez 4 June 30, 2018 3:57 AM

JavaScript Solution:

```
var isAnagram = function(s, t) {
    return (sortStr(s) === sortStr(t));
};
```

Read More

4 Share Reply

SHOW 2 REPLIES

leetcode_deleted_user 275 February 6, 2018 7:21 AM

t=list(t)
t=sorted(t)
s=list(s)
s=sorted(s)
return s==t

Read More

3 Share Reply

SHOW 1 REPLY

knarfamlap 5 February 10, 2019 8:06 PM

```
def isAnagram(self, s, t):

    return sorted(s) == sorted(t)
```

4 Share Reply

SHOW 3 REPLIES

1 2 3 4 5 6 ... 8 9