LeetCode 242 Valid Anagram

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.

Solution

When we get the problem, we first need to check if the characters are belong to alphabets or unicode?

If the string contains only lowercase alphabets, we can define two arrays[26] to save every letter's number, and then compare the two arrays.

Or we define an array[26] to save the number of letters in `s`, and traverse `t` to update the array, in the end to check if array[26] is `0`.

Code

```
bool isAnagram(string s, string t) {
    if(s.size()!=t.size()) return false;
    int arr[26] = {0};
    for(int i=0;i<s.size();++i) {
        char c = s[i];
        arr[c-'a'] ++;
        arr[t[i] - 'a'] --;
    }
```

```
    for(int j=0;j<26;++j) {
        if(arr[j] !=0)
            return false;
    }
    return true;
}
```

`T(n) = O(n), space = O(1) (constant number)`

Follow up

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

If the string contains unicode characters, we can't define an array because we don't know where are the unicode characters' positions. So we can do is just store the characters of inputs and their numbers using hashmap. Here is also two branches. We can have two hashmaps for s and t seperately, and then compare these two hashmaps. Or we have one hashmap for `s`, and update when scan `t`.

```
bool isAnagram(string s, string t) {
    if(s.size() != t.size()) return false;
    unordered_map<wchar_t, int> hamap;
    for(int i=0;i<s.size();++i) {
        if(hsmap.find(s[i])!=hsmap.end())
            hsmap[s[i]] +=1;
        else
            hsmap[s[i]] = 1;
```

```
            if(hsmap.find(t[i]) != hsmap.end())
                hsmap[t[i]] -=1;
            else
                hsmap[t[i]] = -1;
        }
        for(auto it=hsmap.begin(); it!=hsmap.end();++it) {
            if(it->second != 0)
                return false;
        }
        return true;
    }
```

`T(n) = O(n), space = O(n)`

Note

We know in hashmap, find() Time complesity is:

`Average O(1), Worst case O(n)`

However, the worst case is seldom, so we can neglect the worst case when we analyze the time complexity using `find() in hashmap`.

Summary

Cash Model