242. Valid Anagram

- https://leetcode-cn.com/problems/valid-anagram/description/

```
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?
```

```
给定两个字符串 s 和 t ，编写一个函数来判断 t 是否是 s 的字母异位词。
示例 1:
输入: s = "anagram", t = "nagaram"
输出: true
示例 2:
输入: s = "rat", t = "car"
输出: false
说明:
你可以假设字符串只包含小写字母。
进阶:
如果输入字符串包含 unicode 字符怎么办？你能否调整你的解法来应对这种情况？
```

Solutions

1. 两个字符串排序后比对是否一致 O(nlogn)

```
//sort and compare O(nlogn)
class Solution {
public:
    bool isAnagram(string s, string t) {
        if (s.size() != t.size()) { return; }
        sort(s.begin(), s.end());
        sort(t.begin(), t.end());
        return !s.compare(t);
    }
```

```
};
```

2.使用Set去重，再判断字符个数 O(n^2)

- 首先利用set去重排序，然后遍历set里的值，比较这个值在两个字符串里的个数
- 遍历set的过程中，如果个数不一致，则返回false

```
//using set and compare count of characters O(n^2)
class Solution {
public:
    bool isAnagram(string s, string t) {
        if (s.size() != t.size()) { return false; }
        set<char> temp(s.begin(), s.end());
        for (auto &it : temp) {
            if (std::count(s.begin(), s.end(), it) != std::count(t.begin()
, t.end(), it)) {
                return false;
            }
        }
        return true;
    }
};
```

3.使用unordered_map，最后判断是否有非0的Item O(n)– 推荐

```
//using map and compare count of characters O(n)
class Solution {
public:
    bool isAnagram(string s, string t) {
        if (s.size() != t.size()) { return false; }
        unordered_map<char, int> countMap;
        for(int i = 0; i < s.size(); i++) {
            countMap[s[i]]++;
            countMap[t[i]]--;
        }
        return !(std::any_of(countMap.begin(), countMap.end(), []
(const auto &it){ return it.second != 0; }));
    }
};
```

4.使用计数Table O(string.size()) – 推荐

- 建立一个大小为26的计数Table(因为英文小写字母就是26个），Table里的值全部初始化为0。
- 然后遍历这两个字符串，将位置为index(字符 – ’a’)的计数加1或减1。
- 最后判断里面是否有非0的count，或者0的个数是否等于26。

```
//Traversal and using count table O(n)
class Solution {
```

```
public:
    bool isAnagram(string s, string t) {
        if (s.size() != t.size()) { return false; }
        int charCount = 26;
        vector<int> countTable(charCount);
        for(int i = 0; i < s.size(); i++) {
            countTable[s[i] - 'a']++;
            countTable[t[i] - 'a']--;
        }
        //return (std::count(countTable.begin(), countTable.end(), 0) == charCount);
        return !(std::any_of(countTable.begin(), countTable.end(), [](int x){ return x != 0; }));
    }
};
```

5.使用计数Table，两次遍历 O(n) – 效率最高 – 推荐

- 建立一个大小为26的计数Table(因为英文小写字母就是26个），Table里的值全部初始化为0。
- 先遍历第一个字符串，将位置为index(字符 – 'a')的计数加1。
- 然后再遍历第一个字符串，将位置为index(字符 – 'a')的计数减1，同时判断这个位置上的计数如果小于0了，则返回false
- 理论上效率会更高，有可能在第二次遍历时，如果出现计数小于0的，则提前结束遍历。

```
//Traversal and using count table O(n)
class Solution {
public:
    bool isAnagram(string s, string t) {
        if (s.size() != t.size()) { return false; }
        vector<int> countTable(26);
        for (auto &it : s) { countTable[it - 'a']++; }
        for (auto &it : t) {
            if(--countTable[it - 'a'] < 0) { return false; }
        }
        return true;
    }
};
```

Code

```
#include <iostream>
#include <string>
#include <vector>
#include <set>
#include <unordered_map>
#include <algorithm>
using namespace std;
```

```
//sort and compare O(nlogn)
// class Solution {
// public:
//     bool isAnagram(string s, string t) {
//         if (s.size() != t.size()) { return; }
//         sort(s.begin(), s.end());
//         sort(t.begin(), t.end());
//         return !s.compare(t);
//     }
// };
//using set and compare count of characters O(n^2)
// class Solution {
// public:
//     bool isAnagram(string s, string t) {
//         if (s.size() != t.size()) { return false; }
//         set<char> temp(s.begin(), s.end());
//         for (auto &it : temp) {
//             if (std::count(s.begin(), s.end(), it) != std::count(t.begi
n(), t.end(), it)) {
//                 return false;
//             }
//         }
//         return true;
//     }
// };
//using map and compare count of characters O(n)
// class Solution {
// public:
//     bool isAnagram(string s, string t) {
//         if (s.size() != t.size()) { return false; }
//         unordered_map<char, int> countMap;
//         for(int i = 0; i < s.size(); i++) {
//             countMap[s[i]]++;
//             countMap[t[i]]--;
//         }
//         return !(std::any_of(countMap.begin(), countMap.end(), []
(const auto &it){ return it.second != 0; }));
//     }
// };
//Traversal and using count table O(n)
// class Solution {
// public:
//     bool isAnagram(string s, string t) {
//         if (s.size() != t.size()) { return false; }
//         int countSize = 26;
//         vector<int> countTable(countSize);
//         for(int i = 0; i < s.size(); i++) {
//             countTable[s[i] - 'a']++;
//             countTable[t[i] - 'a']--;
```

```
//         }
//         //return (std::count(countTable.begin(), countTable.end(), 0) ==
countSize);
//         return !(std::any_of(countTable.begin(), countTable.end(), [](int x){ return x != 0; }));
//     }
// };
//Traversal and using count table O(n)
class Solution {
public:
    bool isAnagram(string s, string t) {
        if (s.size() != t.size()) { return false; }
        vector<int> countTable(26);
        for (auto &it : s) { countTable[it - 'a']++; }
        for (auto &it : t) {
            if(--countTable[it - 'a'] < 0) { return false; }
        }
        return true;
    }
};
int main() {
    // std::string s = "anagram";
    // std::string t = "nagaram";
    std::string s = "rat";
    std::string t = "car";
    Solution solution;
    string result = solution.isAnagram(s, t) ? "yes!" : "no!";
    std::cout << result << std::endl;
    return 0;
}
```