# 题目

给定两个字符串 s 和 t ，判断它们是否是同构的。

如果 s 中的字符可以按某种映射关系替换得到 t ，那么这两个字符串是同构的。

每个出现的字符都应当映射到另一个字符，同时不改变字符的顺序。不同字符不能映射到同一个字符上，相同字符只能映射到同一个字符上，字符可以映射到自己本身。

示例 1:

输入：s = "egg", t = "add"

输出：true

示例 2：

输入：s = "foo", t = "bar"

输出：false

示例 3：

输入：s = "paper", t = "title"

输出：true

提示：

1 <= s.length <= 5 \* 104

t.length == s.length

s 和 t 由任意有效的 ASCII 字符组成

# 分析

## 方法一：哈希表

要判断两个字符串是否同构，核心思路是建立双向映射关系，确保s中的每个字符唯一映射到t中的某个字符，同时t中的每个字符也唯一被s中的某个字符映射（避免“多对一”或“一对多”的无效映射）。

解题思路：

1、映射关系的双向约束：

- 同构的关键是“一一对应”：s的字符c1必须始终映射到t的字符c2，且c2不能被s的其他字符映射。

- 例如：s="egg"、t="add"中，e→a、g→d，双向映射唯一，故同构；而s="foo"、t="bar"中，o需映射到o和r（一对多），故不同构。

2、用哈希表（数组）存储映射：

- 由于字符是ASCII码（范围0-127），可用大小为128的数组替代哈希表，存储s→t和t→s的映射关系，效率更高。

- 数组map\_s2t：map\_s2t[c] = d表示s中的字符c映射到t中的字符d。

- 数组map\_t2s：map\_t2s[d] = c表示t中的字符d被s中的字符c映射。

3、遍历验证映射一致性：

- 遍历两个字符串的每个字符s\_char和t\_char：

1）若map\_s2t中已存在s\_char的映射，但映射值不等于当前t\_char→ 映射冲突，返回false。

2）若map\_t2s中已存在t\_char的映射，但映射值不等于当前s\_char→ 映射冲突，返回false。

3）若均无映射，建立双向映射（map\_s2t[s\_char] = t\_char，map\_t2s[t\_char] = s\_char）。

- 遍历结束无冲突，返回true。

代码：

class Solution {

public:

bool isIsomorphic(string s, string t) {

// 用大小为128的数组存储ASCII字符的映射关系（初始值设为0，代表未映射）

char map\_s2t[128] = {0}; // s的字符 → t的字符

char map\_t2s[128] = {0}; // t的字符 → s的字符

int n = s.size();

for (int i = 0; i < n; ++i) {

char s\_char = s[i];

char t\_char = t[i];

// 情况1：s\_char已映射，但映射的t字符与当前t\_char不一致

if (map\_s2t[s\_char] != 0 && map\_s2t[s\_char] != t\_char) {

return false;

}

// 情况2：t\_char已映射，但映射的s字符与当前s\_char不一致

if (map\_t2s[t\_char] != 0 && map\_t2s[t\_char] != s\_char) {

return false;

}

// 情况3：均未映射，建立双向映射

if (map\_s2t[s\_char] == 0 && map\_t2s[t\_char] == 0) {

map\_s2t[s\_char] = t\_char;

map\_t2s[t\_char] = s\_char;

}

}

// 遍历结束无冲突，说明同构

return true;

}

};

代码解释

1、映射数组初始化：

用char数组map\_s2t和map\_t2s存储双向映射，大小128覆盖所有ASCII字符，初始值0（ASCII码中`0`是空字符，不会出现在正常输入中，故可作为“未映射”的标记）。

2、遍历验证逻辑：

- 对每个位置的字符对s\_char和t\_char，先检查已有映射是否冲突：若s\_char已映射到其他t字符，或t\_char已被其他s字符映射，直接返回false。

- 若均无映射，建立双向关联，确保后续相同字符的映射一致。

3、效率优化：

数组访问的时间复杂度为O(1)，优于哈希表的平均O(1)（无哈希冲突处理开销），整体遍历仅需O(n)时间，空间复杂度O(1)（数组大小固定为128，与输入长度无关）。

复杂度分析

- 时间复杂度：O(n)，其中n是字符串的长度。仅需遍历字符串一次，每个字符的操作（数组访问、赋值）均为O(1)。

- 空间复杂度：O(1)，映射数组大小固定为128，不随输入规模变化，属于常数级空间。

另一种写法：

class Solution {

public:

bool isIsomorphic(string s, string t) {

unordered\_map<char, char> s2t;

unordered\_map<char, char> t2s;

int len = s.length();

for (int i = 0; i < len; ++i) {

char x = s[i], y = t[i];

if ((s2t.count(x) && s2t[x] != y) || (t2s.count(y) && t2s[y] != x)) {

return false;

}

s2t[x] = y;

t2s[y] = x;

}

return true;

}

};