# 题目

给你一个字符串 s 、一个字符串 t 。返回 s 中涵盖 t 所有字符的最小子串。如果 s 中不存在涵盖 t 所有字符的子串，则返回空字符串 "" 。

注意：

对于 t 中重复字符，我们寻找的子字符串中该字符数量必须不少于 t 中该字符数量。

如果 s 中存在这样的子串，我们保证它是唯一的答案。

示例 1：

输入：s = "ADOBECODEBANC", t = "ABC"

输出："BANC"

解释：最小覆盖子串 "BANC" 包含来自字符串 t 的 'A'、'B' 和 'C'。

示例 2：

输入：s = "a", t = "a"

输出："a"

解释：整个字符串 s 是最小覆盖子串。

示例 3:

输入: s = "a", t = "aa"

输出: ""

解释: t 中两个字符 'a' 均应包含在 s 的子串中，

因此没有符合条件的子字符串，返回空字符串。

提示：

m == s.length

n == t.length

1 <= m, n <= 105

s 和 t 由英文字母组成

进阶：你能设计一个在 o(m+n) 时间内解决此问题的算法吗？

# 分析

## 方法一：滑动窗口

要解决“找到字符串s中涵盖t所有字符的最小子串”的问题，我们可以使用滑动窗口+字符计数的方法。这种方法能够在O(m + n)的时间复杂度内高效求解，其中m和n分别是s和t的长度。

解题思路

1、字符计数初始化：

- 统计字符串t中每个字符的出现次数，存储在target数组中（大小为128，覆盖所有ASCII字符）。

- 初始化滑动窗口的字符计数数组window（同样大小为128），用于记录当前窗口内每个字符的出现次数。

- 记录t中不同字符的数量required，用于判断窗口是否覆盖`t`的所有字符。

2、滑动窗口扩展（右指针）：

- 用right指针遍历s，将当前字符加入窗口，更新window计数。

- 若当前字符是t中的字符，且窗口中该字符的计数等于t中的计数，则formed（已满足的字符种类数）加1。

3、滑动窗口收缩（左指针）：

- 当formed等于required时，说明当前窗口已覆盖t的所有字符。此时尝试移动left指针收缩窗口，以找到更小的有效子串：

- 计算当前窗口的长度，若小于已知最小长度，则更新最小子串的起始索引和长度。

- 移除left指针指向的字符，更新window计数。若该字符是t中的字符且窗口中其计数小于t中的计数，则formed减1，退出收缩阶段。

4、结果提取：

- 遍历结束后，若找到有效子串，则返回该子串；否则返回空字符串。

代码：

class Solution {

public:

string minWindow(string s, string t) {

vector<int> target(128, 0); // 记录t中每个字符的需求数量

int required = 0; // t中不同字符的种类数

// 初始化target和required

for (char c : t) {

if (target[c] == 0) {

required++;

}

target[c]++;

}

vector<int> window(128, 0); // 记录当前窗口中每个字符的数量

int formed = 0; // 当前窗口中已满足需求的字符种类数

int left = 0, right = 0; // 滑动窗口的左右指针

int min\_len = INT\_MAX; // 最小子串的长度

int start = 0; // 最小子串的起始索引

while (right < s.size()) {

char c = s[right];

window[c]++; // 将右指针字符加入窗口

// 若当前字符是t中的字符，且数量已满足需求，更新formed

if (target[c] != 0 && window[c] == target[c]) {

formed++;

}

// 当窗口已覆盖t的所有字符，尝试收缩左指针

while (left <= right && formed == required) {

c = s[left];

// 更新最小子串

int current\_len = right - left + 1;

if (current\_len < min\_len) {

min\_len = current\_len;

start = left;

}

// 移除左指针字符，更新window

window[c]--;

// 若该字符是t中的字符且数量不再满足需求，减少formed

if (target[c] != 0 && window[c] < target[c]) {

formed--;

}

left++; // 左指针右移，收缩窗口

}

right++; // 右指针右移，扩展窗口

}

// 若找到有效子串，返回它；否则返回空字符串

return min\_len == INT\_MAX ? "" : s.substr(start, min\_len);

}

};

解释

1、字符计数数组：target和window数组分别记录t和当前窗口中每个字符的出现次数，利用ASCII码索引实现O(1)的访问和更新。

2、required与formed：required是t中不同字符的总数，formed是当前窗口中满足“计数不少于t中对应字符”的字符种类数。当formed == required时，窗口覆盖t的所有字符。

3、滑动窗口操作：

- 扩展阶段：right指针右移，将字符加入窗口，直到窗口覆盖t的所有字符（formed == required）。

- 收缩阶段：left指针右移，尝试减小窗口 size，同时保持窗口的有效性（formed == required）。每次收缩都检查是否找到更小的有效子串。

4、时间复杂度：O(m + n)，其中m是s的长度，n是t的长度。right和left指针分别遍历s一次（共O(m)），初始化target数组耗时O(n)。

5、空间复杂度：O(1)，两个计数数组大小固定为128，与输入规模无关。