# 题目

给定字符串 S，找出最长重复子串的长度。如果不存在重复子串就返回 0。

示例 1：

输入："abcd"

输出：0

解释：没有重复子串。

示例 2：

输入："abbaba"

输出：2

解释：最长的重复子串为 "ab" 和 "ba"，每个出现 2 次。

示例 3：

输入："aabcaabdaab"

输出：3

解释：最长的重复子串为 "aab"，出现 3 次。

示例 4：

输入："aaaaa"

输出：4

解释：最长的重复子串为 "aaaa"，出现 2 次。

提示：

字符串 S 仅包含从 'a' 到 'z' 的小写英文字母。

1 <= S.length <= 1500

# 分析

## 方法一：二分查找

LeetCode 875. 爱吃香蕉的珂珂（二分查找）

LeetCode LCP 12. 小张刷题计划（二分查找）

LeetCode 1011. 在 D 天内送达包裹的能力（二分查找）

LeetCode 5438. 制作 m 束花所需的最少天数（二分查找）

思路：

字符串的“最长重复子串”有两个关键属性：

1、长度范围可控：最长可能的重复子串长度为 n-1（n 是字符串长度，因为长度为 n 的子串只有 1 个，无法重复），最短为 1（非空子串的最小长度）。

2、验证重复可高效实现：若已知子串长度 L，可通过滑动窗口提取所有长度为 L 的子串，再用哈希表判断是否有重复。

基于这两个属性，我们用二分查找快速缩小“可能的最长长度”，再用滑动窗口 + 哈希表验证该长度是否存在重复子串 —— 避免了“从最长到最短逐一验证”的低效（时间复杂度从 O(n³) 降至 O(n² log n)）。

代码：

class Solution {

public:

int longestRepeatingSubstring(string s) {

int n = s.size();

if (n < 2) return 0; // 长度<2的字符串无重复子串（单个字符无法重复）

// 左边界从1开始（重复子串最小长度为1）

int left = 1, right = n - 1;

int maxLen = 0;

while (left <= right) {

int mid = left + (right - left) / 2;

if (hasRepeatingSubstring(s, mid)) {

maxLen = mid; // 确认存在该长度的重复子串，更新最大长度

left = mid + 1; // 尝试更长的子串

} else {

right = mid - 1; // 该长度无重复，尝试更短的子串

}

}

return maxLen;

}

private:

bool hasRepeatingSubstring(const string& s, int L) {

int n = s.size();

unordered\_set<string> seen;

// 滑动窗口提取所有长度为L的子串（i的范围确保子串不越界）

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

string sub = s.substr(i, L);

if (seen.count(sub)) {

return true; // 找到重复子串

}

seen.insert(sub);

}

return false; // 无重复子串

}

};

## 方法二：后缀数组