# 题目

编写一个算法来判断一个数 n 是不是快乐数。

「快乐数」 定义为：

对于一个正整数，每一次将该数替换为它每个位置上的数字的平方和。

然后重复这个过程直到这个数变为 1，也可能是 无限循环 但始终变不到 1。

如果这个过程 结果为 1，那么这个数就是快乐数。

如果 n 是 快乐数 就返回 true ；不是，则返回 false 。

示例 1：

输入：n = 19

输出：true

解释：

12 + 92 = 82

82 + 22 = 68

62 + 82 = 100

12 + 02 + 02 = 1

示例 2：

输入：n = 2

输出：false

提示：

1 <= n <= 231 - 1

# 分析

要判断一个数是否为“快乐数”，核心是解决两个关键问题：计算一个数的各位数字平方和，以及检测计算过程中是否出现无限循环（若循环则不是快乐数）。常用的循环检测方法有“哈希集合记录”和“快慢指针”，以下分别介绍思路并实现高效解法。

解题思路：

1、核心概念：快乐数的判定逻辑

- 快乐数的定义：反复计算数字的“各位平方和”，若最终能得到1，则是快乐数；若陷入无限循环（即某一结果重复出现），则不是。

- 关键观察：计算过程中若出现重复的数，必然会进入无限循环（因为后续计算会完全重复之前的路径），因此循环检测是核心。

2、两种循环检测方法

方法一：哈希集合（直观易懂）

- 用哈希集合存储每次计算得到的“各位平方和”。

- 每次计算后，若结果为1 → 返回true；若结果已在集合中 → 说明循环，返回false；否则将结果加入集合，继续计算。

方法二：快慢指针（空间优化）

- 借鉴“链表判环”的思路：用两个指针（快指针fast、慢指针slow）同时计算平方和。

- 慢指针每次计算1次平方和，快指针每次计算2次平方和。

- 若存在循环，快指针最终会追上慢指针（两者值相等）；若最终快指针先到达1 → 返回true。

- 优势：无需额外存储空间（空间复杂度O(1)），比哈希集合更高效。

代码实现（快慢指针法，最优解）：

class Solution {

public:

bool isHappy(int n) {

// 快慢指针初始化：慢指针从n开始，快指针从n的平方和开始

int slow = n;

int fast = getSquareSum(n);

// 循环条件：快指针未到1，且快慢指针未相遇（无循环）

while (fast != 1 && slow != fast) {

slow = getSquareSum(slow); // 慢指针走1步（计算1次平方和）

fast = getSquareSum(getSquareSum(fast)); // 快指针走2步（计算2次平方和）

}

// 若快指针到达1，说明是快乐数；否则是循环（快慢指针相遇）

return fast == 1;

}

private:

// 辅助函数：计算一个数的各位数字平方和

int getSquareSum(int num) {

int sum = 0;

while (num > 0) {

int digit = num % 10; // 取当前个位数字

sum += digit \* digit; // 累加平方值

num = num / 10; // 去掉个位数字，处理下一位

}

return sum;

}

};

代码解释：

1、辅助函数getSquareSum

- 功能：计算任意正整数的“各位数字平方和”。

- 逻辑：通过num % 10取个位数字，平方后累加；再通过num / 10去掉个位，循环处理所有位数，直到num变为0。

- 示例：计算19的平方和 → 1² + 9² = 1 + 81 = 82，函数返回82。

2、快慢指针主逻辑

- 初始化：slow从初始值n开始，fast从n的第一次平方和开始（避免初始值相等导致循环不启动）。

- 循环过程：

慢指针每次“走1步”：计算1次平方和。

快指针每次“走2步”：连续计算2次平方和。

若fast先到1 → 说明是快乐数，返回true。

若slow == fast（且fast != 1） → 说明循环，返回false。

复杂度分析

- 时间复杂度：O(log n)。

每次计算平方和时，数字的位数会大幅减少（例如：10^k的数字会变为k\*9²，位数从k+1降至log(k\*81)），因此计算次数与n的位数成对数关系，整体时间复杂度为O(log n)。

- 空间复杂度：O(1)。

仅使用slow、fast、sum等有限变量，无额外存储空间，优于哈希集合的O(log n)空间。