# 题目

给定一个长度为 n 的 0 索引整数数组 nums。初始位置在下标 0。

每个元素 nums[i] 表示从索引 i 向后跳转的最大长度。换句话说，如果你在索引 i 处，你可以跳转到任意 (i + j) 处：

0 <= j <= nums[i] 且

i + j < n

返回到达 n - 1 的最小跳跃次数。测试用例保证可以到达 n - 1。

示例 1:

输入: nums = [2,3,1,1,4]

输出: 2

解释: 跳到最后一个位置的最小跳跃数是 2。

从下标为 0 跳到下标为 1 的位置，跳 1 步，然后跳 3 步到达数组的最后一个位置。

示例 2:

输入: nums = [2,3,0,1,4]

输出: 2

提示:

1 <= nums.length <= 104

0 <= nums[i] <= 1000

题目保证可以到达 n - 1

# 分析

要解决“到达数组最后一个位置的最小跳跃次数”问题，核心思路是贪心策略：通过维护“当前跳跃的最大覆盖范围”和“下一次跳跃的最大覆盖范围”，在不遍历所有可能路径的情况下，找到最少跳跃次数（每一步都选择能覆盖最远位置的跳法）。

解题思路

1、核心观察：

最小跳跃次数的关键是“每一步尽可能跳得远”。例如，从位置i出发，若能跳到i+1、i+2、…、i+nums[i]，则最优选择是跳到“能覆盖更远位置的那个点”（无需关心具体跳哪个点，只需跟踪最远覆盖范围）。

2、关键变量定义：

- jump\_count：当前已完成的跳跃次数（初始为0）。

- current\_end：当前跳跃的最大覆盖边界（初始为0，代表第一次跳跃前的覆盖范围是位置0）。

- max\_reach：截至当前位置，能到达的最远位置（初始为0，动态更新）。

3、贪心遍历逻辑：

遍历数组（不包括最后一个元素，因为到达最后一个元素即终止）：

1）对每个位置i，更新max\_reach = max(max\_reach, i + nums[i])（当前位置能覆盖的最远位置）。

2）若i到达current\_end（当前跳跃的边界）：

- 说明需要进行一次新的跳跃（jump\_count += 1）。

- 将current\_end更新为max\_reach（新跳跃的边界是之前能覆盖的最远位置）。

- 若current\_end已覆盖最后一个元素，可提前终止（优化）。

4、终止条件：

当current\_end覆盖最后一个元素时，jump\_count即为最小跳跃次数。

代码：

class Solution {

public:

int jump(vector<int>& nums) {

int n = nums.size();

if (n == 1) return 0; // 边界：数组长度为1，无需跳跃

int jump\_count = 0; // 已完成的跳跃次数

int current\_end = 0; // 当前跳跃的最大覆盖边界

int max\_reach = 0; // 截至当前位置，能到达的最远位置

// 遍历数组（不包括最后一个元素，到达则终止）

for (int i = 0; i < n - 1; ++i) {

// 更新当前能覆盖的最远位置

max\_reach = max(max\_reach, i + nums[i]);

// 到达当前跳跃的边界，需要进行下一次跳跃

if (i == current\_end) {

jump\_count++; // 跳跃次数+1

current\_end = max\_reach;// 新边界更新为最远覆盖位置

// 优化：若新边界已覆盖最后一个元素，提前终止

if (current\_end >= n - 1) {

break;

}

}

}

return jump\_count;

}

};

代码解释

1、边界处理：若数组长度为1（已在最后一个位置），直接返回0，无需跳跃。

2、遍历与覆盖范围更新：

- 对每个位置i，计算i + nums[i]（从i能跳到的最远位置），并更新max\_reach（确保始终跟踪最远覆盖范围）。

3、跳跃次数更新：

- 当i到达current\_end（当前跳跃的边界），说明当前跳跃已无法覆盖更远位置，必须进行新的跳跃：

jump\_count加1（完成一次跳跃）。

current\_end更新为max\_reach（新跳跃的边界是之前能覆盖的最远位置，确保下一次跳跃尽可能远）。

- 若current\_end已覆盖最后一个元素（current\_end >= n-1），可提前终止循环（无需遍历剩余元素，优化效率）。

复杂度分析

- 时间复杂度：O(n)，其中n是数组长度。仅遍历数组一次，每个元素的操作（更新max\_reach、判断边界）均为O(1)，整体为线性时间。

- 空间复杂度：O(1)。仅使用jump\_count、current\_end、max\_reach三个变量，无额外动态空间开销。