# 题目

你是一个专业的小偷，计划偷窃沿街的房屋，每间房内都藏有一定的现金。这个地方所有的房屋都 围成一圈 ，这意味着第一个房屋和最后一个房屋是紧挨着的。同时，相邻的房屋装有相互连通的防盗系统，如果两间相邻的房屋在同一晚上被小偷闯入，系统会自动报警 。

给定一个代表每个房屋存放金额的非负整数数组，计算你 在不触动警报装置的情况下 ，今晚能够偷窃到的最高金额。

示例 1：

输入：nums = [2,3,2]

输出：3

解释：你不能先偷窃 1 号房屋（金额 = 2），然后偷窃 3 号房屋（金额 = 2）, 因为他们是相邻的。

示例 2：

输入：nums = [1,2,3,1]

输出：4

解释：你可以先偷窃 1 号房屋（金额 = 1），然后偷窃 3 号房屋（金额 = 3）。

偷窃到的最高金额 = 1 + 3 = 4 。

示例 3：

输入：nums = [1,2,3]

输出：3

提示：

1 <= nums.length <= 100

0 <= nums[i] <= 1000

# 分析

要解决“环形房屋偷窃”问题，核心思路是将环形约束转化为线性问题：利用房屋环形排列的特性（首末房屋相邻，不可同时偷窃），将原问题拆分为两个无环形约束的线性偷窃问题，再取两者的最大值。

解题思路

1、环形约束的突破：

环形房屋的关键限制是“首屋和末屋不可同时偷窃”，因此可将问题拆分为两种互斥的线性场景：

- 场景1：不偷末屋，仅考虑偷窃范围 [0, n-2]（n为房屋总数）。

- 场景2：不偷首屋，仅考虑偷窃范围 [1, n-1]。

最终答案为两种场景的最大偷窃金额的最大值（确保不违反环形约束）。

2、线性房屋的动态规划（DP）解法：

对于任意线性范围 [start, end]，最大偷窃金额可通过优化后的DP求解：

- 状态定义：无需用数组存储所有状态，仅需两个变量：

prev\_prev：表示前两间房屋的最大偷窃金额（对应 dp[i-2]）。

prev：表示前一间房屋的最大偷窃金额（对应 dp[i-1]）。

- 状态转移：对于当前房屋 i，最大金额为“不偷当前屋（取prev）”或“偷当前屋（取prev\_prev + nums[i]）”的较大值，即curr = max(prev, prev\_prev + nums[i])。

- 空间优化：用两个变量替代DP数组，空间复杂度从O(n)降至O(1)。

3、边界处理：

- 若只有1间房屋（n=1），直接返回该房屋金额（无环形约束）。

- 若线性范围为空（如 start > end），返回0；若范围仅1间房屋，返回该房屋金额。

代码：

class Solution {

public:

int rob(vector<int>& nums) {

int n = nums.size();

if (n == 1) {

return nums[0]; // 边界：仅1间房屋，直接返回金额

}

// 拆分为两个线性场景，取最大值

return max(robLinear(nums, 0, n-2), robLinear(nums, 1, n-1));

}

private:

// 计算线性范围 [start, end] 内的最大偷窃金额

int robLinear(vector<int>& nums, int start, int end) {

// 边界1：范围为空（如n=2时，场景1的end=0，start=1，start>end）

if (start > end) {

return 0;

}

// 边界2：范围仅1间房屋

if (start == end) {

return nums[start];

}

// 初始化前两间房屋的最大金额

int prev\_prev = nums[start]; // 对应 i = start（第1间）

int prev = max(nums[start], nums[start+1]); // 对应 i = start+1（第2间）

// 遍历从第3间房屋到最后1间（i = start+2 到 end）

for (int i = start + 2; i <= end; ++i) {

int curr = max(prev, prev\_prev + nums[i]); // 当前房屋的最大金额

prev\_prev = prev; // 更新前两间为前一间

prev = curr; // 更新前一间为当前

}

return prev; // 最终prev为范围[start, end]的最大金额

}

};

复杂度分析

- 时间复杂度：O(n)，其中n是房屋数量。两个线性场景的遍历总次数为O(n)，无冗余操作。

- 空间复杂度：O(1)，仅使用prev\_prev、prev等固定变量，无额外动态空间开销。