# 198.打家劫舍 - 陈洋

🏷️动态规划数组

**题目描述：**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，影响你偷窃的唯一制约因素就是**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**不触动警报装置**的情况下 ，一夜之内能够偷窃到的**最高金额**。

**示例1：**

输入：[1, 2, 3, 1] 输出：4 解释：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2：**

输入：[2, 7, 9, 3, 1] 输出：12 解释：偷窃 1 号房屋 (金额 = 2), 偷窃 3 号房屋 (金额 = 9)，接着偷窃 5 号房屋 (金额 = 1)。 偷窃到的最高金额 = 2 + 9 + 1 = 12 。

💡 分析：

* 1. 考虑最简单的情况，即只有**一间房屋**，那么偷窃的**最高金额**就是**该房间的金额**

* 1. 考虑**两间房**的情况，能偷窃的**最高金额**就是两间房**金额最高的一间的金额**

* 1. 如果房间数**大于两间，**对于**第 i 间房**，得分情况讨论**：**

* + 1. 如果选择**偷窃**第 i 间房，根据题目要求，就不能偷窃第 i-1 间房，那么此时最高金额就为**偷窃前 i-2 间房和第 i 间房的金额之和**

* + 1. 如果选择不偷窃第 i 间房，那么最高金额就为 **偷窃前 i -1 间房的金额之和**

* 所以，最高金额就是这两种方案得到的**最大值**，即偷窃前 i 间房能得到的最大金额数

🏆**动态规划：**

如果使用 dp[i] 来表示前 i 间**（从 0 开始）**房能够偷窃的最高金额，那么**状态转移方程**可为：

dp[i] = max(dp[i-2] + money[i], dp[i-1])

**初始化为：**

dp[0] = money[0] -> 表示只有一间房

dp[1] = max(money[0], money[1]) -> 选择两间房金额最高的一间

**算法伪代码：**

**变量说明：**

nums[] - 存储每个房间的金额

length - 存储 nums 数组的长度

dp[i] - 表示前 i 间**（从 0 开始）**房能够偷窃的最高金额

**输入：**

nums[]

**输出：**

能偷窃的最大金额

**处理：**

Step1: 记录序列的长度，存储在 length 变量中

Step2: 讨论简单情况，进行初始化（只有一间房 or 只有两间房）

- 只有一间房，最高金额为该房间的金额数 输出 nums[0]

- 两间房，选择两间房金额最高的一间的金额输出 输出 max(nums[0], nums[1])

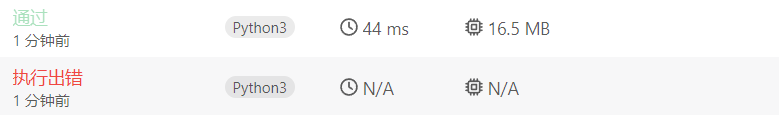
Step3: for 循环填补状态转移方程

Step4: 输出 dp[length-1] 即为最大金额数

🏅**Python代码**

class Solution:  
 def rob(self, nums: List[int]) -> int:  
 length = len(nums)  
 if length == 1:  
 return nums[0]  
 if length == 2:  
 return max(nums[0], nums[1])  
 dp = [0] \* length  
 dp[0],dp[1] = nums[0],max(nums[0], nums[1])  
 for i in range(2, length):  
 dp[i] = max(dp[i-2] + nums[i], dp[i-1])  
 return dp[length-1]

**提交结果：**



🔬**时间复杂度分析：**

一重 for 循环，执行 n-2 次，所以时间复杂度为 **O(n)**