# 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，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 。

**提示：**

0 <= nums.length <= 100

0 <= nums[i] <= 400

# 分析

## 方法一：动态规划

**思路：**

**代码：**

class Solution {

public:

int rob(vector<int>& nums) {

if(nums.empty())return 0;

int size = nums.size();

if(1 == size)return nums[0];

vector<int> dp(size,0);

dp[0] = nums[0];

dp[1] = max(nums[0],nums[1]);

for(int i=2;i<size;i++)

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

return dp[size-1];

}

};

上述方法使用了数组存储结果。考虑到每间房屋的最高总金额只和该房屋的前两间房屋的最高总金额相关，因此可以使用滚动数组，在每个时刻只需要存储前两间房屋的最高总金额。

class Solution {

public:

int rob(vector<int>& nums) {

if (nums.empty()) {

return 0;

}

int size = nums.size();

if (size == 1) {

return nums[0];

}

int first = nums[0], second = max(nums[0], nums[1]);

for (int i = 2; i < size; i++) {

int temp = second;

second = max(first + nums[i], second);

first = temp;

}

return second;

}

};

**复杂度分析：**

时间复杂度：O(n)，其中n是数组长度。只需要对数组遍历一次。

空间复杂度：O(1)。使用滚动数组，可以只存储前两间房屋的最高总金额，而不需要存储整个数组的结果，因此空间复杂度是O(1)。