# 198 House Robber

2018年4月4日 15:07

## Question:

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, the only constraint stopping you from robbing each of them is that adjacent houses have security system connected and **it will automatically contact the police if two adjacent houses were broken into on the same night**.
Given a list of non-negative integers representing the amount of money of each house, determine the maximum amount of money you can rob tonight **without alerting the police**.

来自 <https://leetcode.com/problems/house-robber/description/>

你是一个专业的强盗，计划抢劫沿街的房屋。每间房都藏有一定的现金，阻止你抢劫他们的唯一的制约因素就是相邻的房屋有保安系统连接，**如果两间相邻的房屋在同一晚上被闯入，它会自动联系警方**。
给定一个代表每个房屋的金额的非负整数列表，确定你可以在**没有提醒警方的情况下**抢劫的最高金额。

## Solution for Python3:

```python
class Solution1:
    def rob(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        if not nums:
           return 0
        if len(nums) == 1:
           return nums[0]
        nums[1] = max(nums[0], nums[1]);
        for i in range(2, len(nums)):
           nums[i] = max(nums[i - 2] + nums[i], nums[i - 1]);
        return nums[-1]


# Based on the recursive formula:
    # f(0) = nums[0]
    # f(1) = max(num[0], num[1])
    # f(k) = max( f(k-2) + nums[k], f(k-1) )

class Solution2:
    def rob(self, nums):
        """
        :type nums: List[int]
        :rtype: int
        """
        last, now = 0, 0
        for i in nums:
           last, now = now, max(last + i, now)
        return now
```

## Solution for C++:

```cpp
class Solution1 {
public:
    int rob(vector<int>& nums) {
        if (nums.empty()) {
            return 0;
        }
        if (nums.size() == 1) {
            return nums[0];
        }
        nums[1] = max(nums[0], nums[1]);
        for (int i = 2; i < nums.size(); i++) {
            nums[i] = max(nums[i-2] + nums[i], nums[i-1]);
        }
        return nums[nums.size() - 1];
    }
};

class Solution2 {
public:
    int rob(vector<int>& nums) {
        int last = 0, now = 0, t = 0;
        for (int i : nums) {
            t = now,
            now = max(last + i, now);
            last = t;
        }
        return now;
    }
};
```

## Appendix:

**动态规划类问题：计算到达每一步所能得到的最优值。**
**到达最后一步时就得到全局最优值。**