

PS：如果你是小偷，你怎样才能偷最多的钱呢？

在 LeetCode 官网中看到了这样的评论

 00000

我们是好孩子, 不偷好不好



```
class Solution {  
    public int rob(int[] nums) {  
        return 0;  
    }  
}
```

金刀表示世上还是好人多，这位好孩子虽然输了题目，甚至在面试中挂了，但是但是赢了人生哈哈。 甚至有网友表示

 cuidong

@00000 哈哈，有觉悟，建议保研

 知春路金刀

 最后的星辰  

@00000 不仅不偷，咱还得捐，建议返回负数。

题目介绍^[1]

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 。



微信搜一搜

知春路金刀

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。

题目解答

这题粗略一看，估计又是一道动态规划题型。有一句话说得好，万事开头难，在动态规划题型中，找出子问题是最关键的，毕竟全局解是一步一步求解子问题来得到的。如果子问题是什么都不知道，那咋求出全局解呢。



寻找子问题

原问题:从第0到第N个房子中能偷取到的最大的金额



我们先来看看如何寻找子问题。题目的大意是求解「从 N 个房子中能偷取到的最大的金额」，这个问题是不是可以拆成 N 个子问题？

- 从 1 个房子中能偷取到的最大的金额是多少？
- 从 2 个房子中能偷取到的最大的金额是多少？
- ...
- 从 N-1 个房子中能偷取到的最大的金额是多少？
- 从 N 个房子中能偷取到的最大的金额是多少？



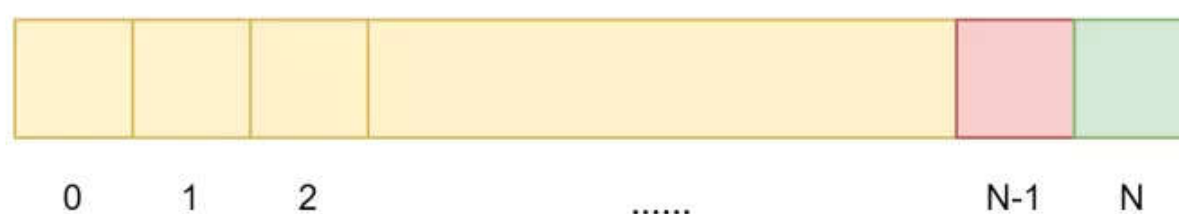
子问题:从第0到第N-1 个房子中能偷取到的最大的金额



注意这题有一个限制就是「小偷不能连续偷两个相邻的房子」，所以小偷经过一个房子时，他需要思考偷还是不偷，这样每一个子问题又可以继续拆解变成 $2N$ 个子问题：

- 从 1 个房子中能偷取到的最大的金额是多少？
 - 不偷第 1 个房子时，能偷取到的最大的金额是多少？
 - 偷第 1 个房子时，能偷取到的最大的金额是多少？
- 从 2 个房子中能偷取到的最大的金额是多少？
 - 不偷第 2 个房子时，能偷取到的最大的金额是多少？
 - 偷第 2 个房子时，能偷取到的最大的金额是多少？
-
- 从 N-1 个房子中能偷取到的最大的金额是多少？
 - 不偷第 N-2 个房子时，能偷取到的最大的金额是多少？
 - 偷第 N-2 个房子时，能偷取到的最大的金额是多少？
- 从 N 个房子中能偷取到的最大的金额是多少？
 - 不偷第 N-1 个房子时，能偷取到的最大的金额是多少？
 - 偷第 N-1 个房子时，能偷取到的最大的金额是多少？

拆解子问题:不偷第N-1个房子时，从第0到第N-1个房子中能偷取到的最大的金额是多少？



拆解子问题:偷第N-1个房子时，从第0到第N-1个房子中能偷取到的最大的金额是多少？



确定状态转移方程

子问题已经确定出来了，那么如果我们知道了「从 N-1 个房子中能偷取到的最大的金额」，那么我们如何根据这个子问题来算出「从 N 个房子中能偷取到的最大的金额」原问题呢？

小偷为了能偷最多的钱，自学了编程然后搞了两个数组 `steal` 和 `not_steal`，`steal[k]`记录小偷偷了第 `K` 个房子时能获取到的最多的钱，`not_steal[k]`记录小偷不偷第 `K` 个房子时能获取到的最多的钱，小偷每到达一个房子的时候，都会去更新 `steal[k]`和 `not_steal[k]`，当小偷来到了第 `K` 个房子的时心里可能这么想：

- 如果第 `K-1` 个房子已经偷了，那第 `K` 个房子就不能偷了，不然该报警了。
 - 所以小偷决定不偷第 `K` 个房子，并记录下当前不偷第 `K` 个房子时总共能获取到的最大的金额为 $not_steal[k] = \max(steal[k-1], not_steal[k-1])$ 。
 - 「解释」：既然不能偷第 `K` 个房子了，那小偷可以偷第 `K-1` 一个房子(`steal[k-1]`)，也可以不偷第 `K-1` 个房子(`not_steal[k-1]`)，小偷当然要选择 `steal[k-1]`和 `not_steal[k-1]`这两个数其中最大的一个。
- 如果第 `K-1` 个房子没有偷，那可以偷第 `K` 个房子也可以不偷第 `K` 个房子。
 - 小偷决定不偷第 `K` 个房子，并记录下当前不偷第 `K` 个房子时总共能获取到的最大的金额同上。
 - 小偷决定偷第 `K` 个房子，那么小偷当前偷第 `K` 个房子时总共能获取到的最大的金额为 $steal[k] = not_steal[k-1] + nums[k]$ 。
 - 「解释」：既然决定偷第 `K` 个房子了，那小偷不可以偷第 `K-1` 个房子(`not_steal[k-1]`)，所以小偷偷完第 `K` 个房子(`steal[k]`)后获取到的最大金额为当前房屋价值再加上 `not_steal[k-1]`。

状态转移方程就确定下来了：

```
not_steal[k] = max(steal[k-1], not_steal[k-1])

steal[k] = not_steal[k-1] + nums[k]
```

最后只需要取`not_steal[k]` 和`steal[k]` 中最大的一个就是我们要的答案。 状态转移方程在动态规划里是最核心的概念，有了状态转移方程，那么就已经有了题目的答案了，剩下的就交给代码来实现了。

原始数组

2	7	9	3	1
---	---	---	---	---

知春路金刀

$steal[0] = nums[0] = 2$

steal	2				
not_steal	0				

$not_steal[0] = 0$

方法一：一维动态规划

代码实现

```
class Solution {
    public int rob(int[] nums) {
        int[][] dp = new int[nums.length+1][2];
        for (int i = 1; i < dp.length; i++) {
            dp[i][0] = dp[i-1][1] + nums[i-1]; // 0 - 表示steal[k]
            dp[i][1] = Math.max(dp[i-1][0], dp[i-1][1]); // 1 - 表示not_steal[k]
        }
        return Math.max(dp[nums.length][0], dp[nums.length][1]);
    }
}
```

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： $O(n)$ 。

方法二：优化动态规划

上面的一维动态规划解法使用了一个 `dp` 数组，我们仔细观察可以发现，计算 `dp[i]` 的状态只取决于 `dp[i-1]` 的状态，所以我们可以用两个临时变量 `steal` 和 `not_steal` 来代替 `dp[i-1][0]` 和 `dp[i-1][1]` 中的值。



微信搜一搜

知春路金刀

代码实现

```
class Solution {
    public int rob(int[] nums) {
        if (nums == null || nums.length == 0) {
            return 0;
        }
        int steal = nums[0], not_steal = 0;
        for (int i = 1; i < nums.length; i++) {
            int new_steal = not_steal + nums[i];
            int new_not_steal = Math.max(steal, not_steal);
            steal = new_steal;
            not_steal = new_not_steal;
        }
        return Math.max(steal, not_steal);
    }
}
```

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)。

总结

算法好的人，当小偷偷的钱都是最多的，你说算法重要不重要？？？



其他

「图解大厂面试高频算法题」专题文章主旨是：根据二八法则的原理，以付出 20%的时间成本，获得 80%的刷题的收益，让那些想进互联网大厂或心仪公司的人少走些弯路。



微信搜一搜

知春路金刀

本专题还在持续更新 ing~ 所有文章、图解和代码全部是金刀亲手完成。内容全部放在了[github](#)^[2]和[gitee](#)^[3]方便小伙伴们阅读和调试，另外还有更多小惊喜等你发现~

如果你喜欢本篇文章，PLZ 一键三连（关注、点赞、在看）。

参考资料

- [1] **原题链接:**
<https://leetcode-cn.com/problems/house-robber/>
- [2] **github:**
<https://github.com/goldknife6>
- [3] **gitee:**
<https://gitee.com/goldknife6>



微信搜一搜

知春路金刀