1. 题目
2. 题干

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。

1. 示例

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 。

二． 题解

（一）思路

从题目来看，很明显又是一道动态规划的题目。这种要求从一堆数中求取最大的和的问题都是用动态规划来解决。题目要求不能使用相连的数，对于动态规划的题目，我们总是要思考，如何从之前的状态转移到现在的状态。设f[n]为存放前n个数之中能够偷窃到最高金额的数组，对于前k个数，如何得到最高金额，这就有两种方式，一种是选择第k个数，那么第k-1个数就不能选了；另一种方式是不选择第k个数，那么第k-1个数就能够选了。在这两种方式中选择最大值就是前k个数中最高金额。

状态转移方程为：

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

边界条件是只有一个或者两个数时：

f[0] = nums[0];

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

（二）代码实现

Java：

class Solution {

    public int rob(int[] nums) {

        int n = nums.length;

        if(nums == null || n == 0){

            return 0;

        }

        if(n==1){

            return nums[0];

        }else if(n == 2){

            return Math.max(nums[0], nums[1]);

        }

        int[] dp = new int[n];

        dp[0] = nums[0];

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

        for(int i =2; i < n; ++i){

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

        }

        Arrays.sort(dp);

        return dp[n-1];

    }

}