Leetcode\_198\_HouseRobber\_打家劫舍\_Easy

# Leetcode\_198\_HouseRobber\_打家劫舍\_Easy

## 问题介绍

\* 难度:Easy

\* https://leetcode.com/problems/house-robber/description/

\* 题目介绍

\* 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.

\*

\* Example 1:

\* Input: [1,2,3,1] Output: 4

\* Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).

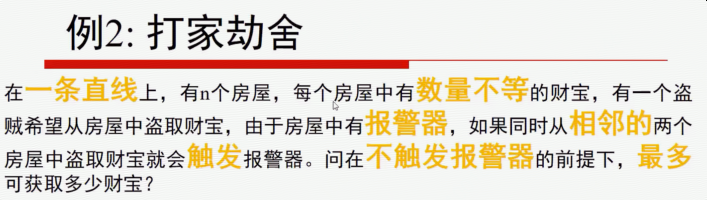
\* Total amount you can rob = 1 + 3 = 4.

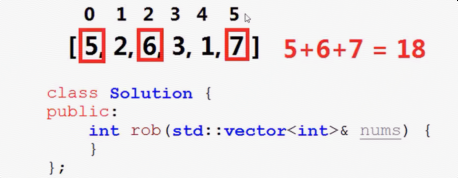
\* Example 2:

\* Input: [2,7,9,3,1] Output: 12

\* Explanation: Rob house 1 (money = 2), rob house 3 (money = 9) and rob house 5 (money = 1).

\* Total amount you can rob = 2 + 9 + 1 = 12.





## 思路分析

\* 思路分析：

\* 方法1：

\* 自变量n是数组nums的最大下标；dp(n)为最大下标为n情况下抢夺的最大钱数；

\* dp(n)有两种情况，一是选择抢nums[n]，则dp(n)=dp[n-2]+nums[n];

\* 二是不选择抢多nums[n],则dp(n)=dp[n-1];

\* 因此状态转移方程为：dp(n) = max{dp[n-2]+nums[n],dp[n-1]}; n>=2;

\* dp(0) = nums[0];

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

\* 方法2：

\* 这种思路是自己考虑的：

\* 如果nums数组最大下标为n，则一定要抢到n的话，对应抢到n的情况有两种：

\* 一是在抢到了n-2的基础上，二是在抢夺了n-3的基础上；

\* 即一定要抢夺n房子，则最大钱数为：dp[i] = max(dp[i-2],dp[i-3])+nums[i];

\* 这时候，相当于更新之后的nums存放的是必须抢夺该房子对应的最大钱数。

\* 最终最大钱数有两种情况：一是抢了n房子，二是没有抢n房子。

\* 返回两者最大值即可：max(dp[nums.length-1],dp[nums.length-2])。

\*

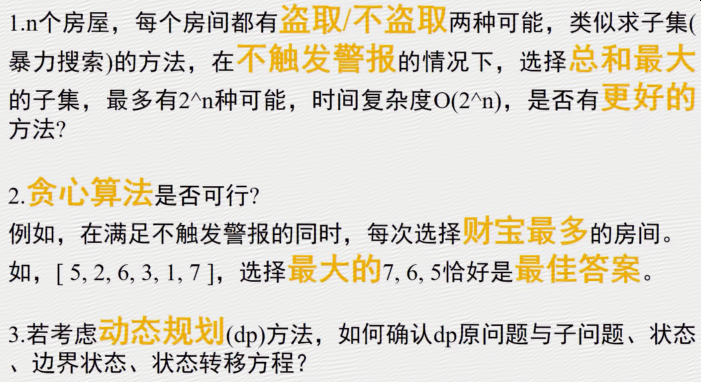
\* 两种方法的区别：方法1的dp[n]中存放的是数组最大索引为n时对应的抢夺的最大钱数，

\* 结果返回dp[n]即可；

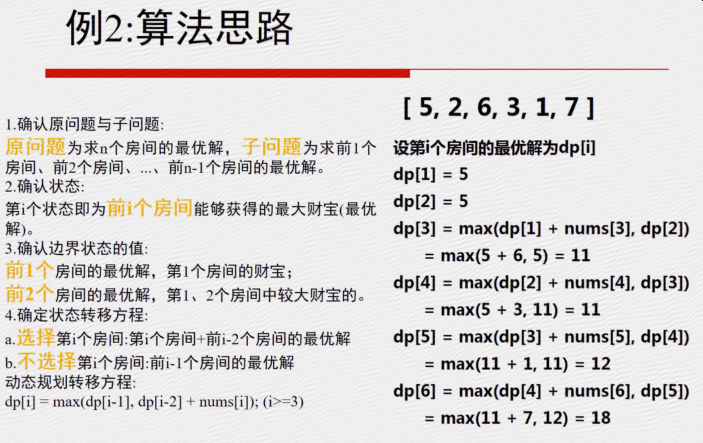
\* 方法2的dp[n]中存放的是在一定抢到nums[n]的情况下，最大的钱数，

\* 最后返回dp[n-1]和dp[n]的最大值即可。

思考：







## Java代码

/\*\*

\* 方法1：dp数组存放的就是当前数组长度下的最大钱数

\*/

public int rob1(int[] nums) {

if(nums == null|| nums.length == 0) return 0;

if(nums.length == 1) return nums[0];

if(nums.length == 2) return max(nums[0],nums[1]);

int[] dp = new int[nums.length];//存放对应最大钱数

dp[0] = nums[0];

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

for(int i = 2;i < nums.length;i++){

dp[i] = max(dp[i-2]+nums[i],dp[i-1]);//状态转移

}

return dp[nums.length-1];

}

public int max(int a,int b){//或者直接使用Math.max方法

return a>b?a:b;

}

/\*\*

\* 方法2：dp存放必须抢该房子的情况下的最大钱数。

\*/

public int rob2(int[] nums) {

if(nums == null|| nums.length == 0) return 0;

if(nums.length == 1) return nums[0];

if(nums.length == 2) return max(nums[0],nums[1]);

if(nums.length == 3) return max(nums[0]+nums[2],nums[1]);

int[] dp = new int[nums.length];

dp[0] = nums[0];

dp[1] = nums[1];

dp[2] = nums[0]+ nums[2];

for(int i = 3;i < nums.length;i++){

dp[i] = max(dp[i-2],dp[i-3])+nums[i];

}

return max(dp[nums.length-1],dp[nums.length-2]);

}

/\*\*

\* 不开辟dp数组

\*/

/\* public int **rob**(int[] nums) {

if(nums == null|| nums.length == 0) return 0;

if(nums.length == 1) return nums[0];

if(nums.length == 2) return nums[0]>nums[1]?nums[0]:nums[1];

if(nums.length == 3) return (nums[0]+nums[2])>nums[1]?(nums[0]+nums[2]):nums[1];

nums[1] = nums[0]>nums[1]?nums[0]:nums[1];

nums[2] = (nums[0]+nums[2])>nums[1]?(nums[0]+nums[2]):nums[1];

for(int i = 3;i < nums.length;i++){

nums[i] += Math.max(nums[i-2],nums[i-3]);

}

return nums[nums.length-1]>nums[nums.length-2]?nums[nums.length-1]:nums[nums.length-2];

}\*/

/\*\*

\* 不开辟dp数组，在nums数组基础上仍然可以存储。

\*/

/\*

public int rob2(int[] nums) {

if(nums == null|| nums.length == 0) return 0;

if(nums.length == 1) return nums[0];

if(nums.length == 2) return max(nums[0],nums[1]);

if(nums.length == 3) return max(nums[0]+nums[2],nums[1]);

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

nums[2] = max(nums[0]+nums[2],nums[1]);

for(int i = 3;i < nums.length;i++){

nums[i] += max(nums[i-2],nums[i-3]);

}

return max(nums[nums.length-1],nums[nums.length-2]);

}\*/

