House Robber II

Question body:

After robbing those houses on that street, the thief has found himself a new place for his thievery so that he will not get too much attention. This time, all houses at this place are arranged in a circle. That means the first house is the neighbor of the last one. Meanwhile, the security system for these houses remain the same as for those in the previous street.

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.

House Robber(A standard dynamic programming 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.

this can be solved using the reoccurrence relationship:

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

Having the house arranged in a circle impose a constraint to the problem which is:

first and last house can not be picked at the same time. Thus, to deal with this situation, we can use dynamic programming on the first n-1 house. This correspond to the case last house is not picked. Then, we can use dynamic programming again on the 2-n house. This corresponds to the case first house is not picked. Then, we just need to compare the two result and return the larger one.

Solution

public class Solution {

public int rob(int[] nums) {

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

return 0;

if(nums.length==1)

return nums[0];

int[] dp\_1 = new int[nums.length-1];

int[] dp\_2 = new int[nums.length-1];

if(nums.length==2){

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

}

dp\_1[0] = nums[1];

dp\_1[1]=Math.max(nums[1], nums[2]);

dp\_2[0]=nums[0];

dp\_2[1]=Math.max(nums[0], nums[1]);

for(int i=2; i<nums.length-1; i++){

dp\_1[i] = Math.max(dp\_1[i-2]+nums[i+1], dp\_1[i-1]);

}

for(int i=2; i<nums.length-1; i++){

dp\_2[i] = Math.max(dp\_2[i-2]+nums[i], dp\_2[i-1]);

}

return Math.max(dp\_1[nums.length-2],dp\_2[nums.length-2]);

}

}