213. House Robber II

Medium 2512 60 Add to List Share

You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed. All houses at this place are **arranged in a circle.** That means the first house is the neighbor of the last one. Meanwhile, adjacent houses have a 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 `nums` representing the amount of money of each house, return *the maximum amount of money you can rob tonight* ***without alerting the police***.

Example 1:

```
Input: nums = [2,3,2]
Output: 3
Explanation: You cannot rob house 1 (money = 2) and then rob
house 3 (money = 2), because they are adjacent houses.
```

Example 2:

```
Input: nums = [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 3:

```
Input: nums = [0]
Output: 0
```

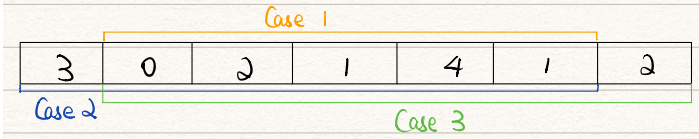
Constraints:

- `1 <= nums.length <= 100`
- `0 <= nums[i] <= 1000`

Definition:
rob[i]: the maximum money you can rob tonight if you rob ith house

Base case:
rob[0] = nums[0]
rob[1] = max(nums[0], nums[1])

Recurrence relation:
rob[i] = max(rob[i-1], rob[i-2] + nums[i])



Compare case 1 and case 2.

Time complexity: O(n)
Space complexity: O(1)

```cpp
class Solution {
public:
    int rob(vector<int>& nums) {
        if (nums.size() == 0) {
            return 0;
        } else if (nums.size() == 1) {
            return nums[0];

        } else if (nums.size() == 2) {
            return max(nums[0], nums[1]);
        }

        vector<int> nums1(nums.begin(), nums.end() - 1);
        vector<int> nums2(nums.begin() + 1, nums.end());

        return max(rob_line(nums1), rob_line(nums2));
    }

private:
    int rob_line(vector<int> nums) {
        // rob[0] = nums[0]
        int rob_i_prev = nums[0];
        // rob[1] = nums[1]
        int rob_i = max(nums[0], nums[1]);

        for (auto it = nums.begin() + 2; it != nums.end(); it++) {
            int rob_i_temp = rob_i;
            // rob[i] = max(rob[i-1], rob[i-2] + nums[i])
            rob_i = max(rob_i_temp, rob_i_prev + *it);
            // rob[i-2] = rob[i-1]
            rob_i_prev = rob_i_temp;
        }

        return rob_i;
    }
};
```