Dynamic Programming

Fibonacci / 1-Dimensional style

Min Cost Climbing Stairs (LC 746)

746. Min Cost Climbing Stairs
Easy

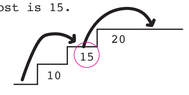
You are given an integer array cost where `cost[i]` is the cost of ith step on a staircase. Once you pay the cost, you can either climb one or two steps.

You can either start from the step with index 0, or the step with index 1.

Return the minimum cost to reach the top of the floor.

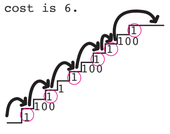
```
Example 1:

Input: cost = [10,15,20]
Output: 15
Explanation: You will start at index 1.
- Pay 15 and climb two steps to reach the top.
The total cost is 15.
```




```
Example 2:

Input: cost = [1,100,1,1,1,100,1,1,100,1]
Output: 6
Explanation: You will start at index 0.
- Pay 1 and climb two steps to reach index 2.
- Pay 1 and climb two steps to reach index 4.
- Pay 1 and climb two steps to reach index 6.
- Pay 1 and climb one step to reach index 7.
- Pay 1 and climb two steps to reach index 9.
- Pay 1 and climb one step to reach the top.
The total cost is 6.
```



```
import collections
class Solution:
    def minCostClimbingStairs(self, cost: List[int]) -> int:
        for i in range(len(cost) - 3, -1, -1):
            cost[i] += min(cost[i + 1], cost[i + 2])

        return min(cost[0], cost[1])
```

example 2:
cost = [1,100,1,1,1,100,1,1,100,1]

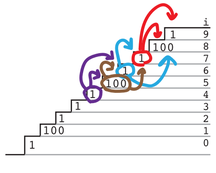
```
len(cost) -> 10
for i in range(7, -1, -1)
```

i = 7 { cost[i] = min(cost[i+1], cost[i+2]) [1,100,1,1,1,100,1,2,100,1]

i = 6 { cost[i] = min(cost[i+1], cost[i+2]) [1,100,1,1,1,100,3,2,100,1]

i = 5 { cost[i] = min(cost[i+1], cost[i+2]) [1,100,1,1,1,102,3,2,100,1]

i = 5 { cost[i] = min(cost[i+1], cost[i+2]) [1,100,1,1,4,102,3,2,100,1]



cost[0] because it records the smallest (minimum) between cost[i+1], cost[i+2] is the smallest possible value

House Robber (LC 198)

198. House Robber
Medium

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 systems connected and it will automatically contact the police if two adjacent houses were broken into on the same night.

Given an integer array 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:

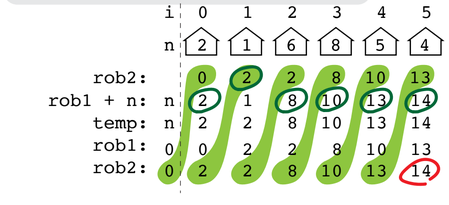
Input: nums = [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.

Example 2:

Input: nums = [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..

```
Q1: houses- [2, 1, 1, 2];        solution- 4
Q2: houses- [2, 7, 9, 3, 1];     solution- 12
Q3: houses- [2, 1, 6, 8, 5, 4]; solution- 14
Q4: houses- [2, 1, 8, 6, 5, 4]; solution- 15
```

```
class Solution:
    def rob(self, nums: List[int]) -> int:
        rob1, rob2 = 0, 0
        for n in nums:
            temp = max(n + rob1, rob2)
            rob1 = rob2
            rob2 = temp
        return rob2
```



1. declare two variables that keep track of previous values
2. alternate the values between the two variables to ensure they are not adjacent

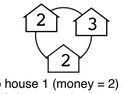
House Robber II (LC 213)

213. House Robber II
Medium

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.

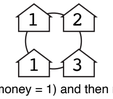
Given an integer array nums representing the amount of money of each house, return the maximum amount of money you can rob tonight without alerting the police.

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:



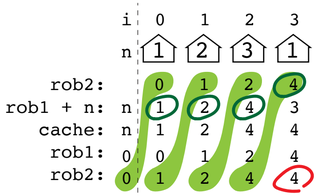
Input: nums = [1,2,3,1]
Output: 4
Explanation: Rob house 1 (money = 1) and then rob house 3 (money = 3).
Total amount you can rob = 1 + 3 = 4.

Example 3:

Input: nums = [1,2,3]
Output: 3

```
class Solution:
    def rob(self, nums: List[int]) -> int:
        return max(nums[0], self.analysis(nums[1:]), self.analysis(nums[:-1]))

    def analysis(self, nums):
        rob1, rob2 = 0, 0
        for n in nums:
            cache = max(rob1 + n, rob2)
            rob1 = rob2
            rob2 = cache
        return rob2
```




```
i:0 1 2 3
nums[0]    = [1]
nums       = [1,2,3,1]
nums[1:]   =   [2,3,1]
nums[:-1]  = [1,2,3]
```

nums[1:]=[2,3,1]

nums[:-1]=[1,2,3]

perform the 'house robber' algorithm on two parts of the nums list, that either (1) excludes the first number, or (2) excludes the last number