Documentation for House Robber Problem

Problem Statement:

- 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 are broken into on the same night.

- Given an integer array nums representing the amount of money in each house, the goal is to return the maximum amount of money you can rob without alerting the police by robbing two adjacent houses.

Constraints:

- 1 <= nums.length <= 100: The number of houses ranges from 1 to 100.
- 0 <= nums[i] <= 400: Each house contains a non-negative integer representing the amount of money stashed in that house, ranging from 0 to 400.

Example 1:

- **Input:** nums = [1, 2, 3, 1]
- **Output:** 4
- **Explanation:** Rob house 1 (money = 1) and then rob house 3 (money = 3). The 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). The total amount you can rob = 2 + 9 + 1 = 12.

Approach:

This problem can be solved using dynamic programming by observing the following insights:

1. **Decision Problem:**
   - ***For each house i, you have two options:***
     - *Rob the current house:* This means you can't rob the previous house (i-1), but you can add the current house's value to the money robbed from house i-2.
     - *Skip the current house:* This means you simply take the maximum amount you have robbed up to house i-1.

2. **Dynamic Programming Table (dp):**

- Let dp[i] represent the maximum amount of money you can rob from the first i houses.
- ***The recurrence relation will be:***

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

  - ***This formula states that for any house i, the maximum money you can rob is the maximum of either:***
    - Skipping house i and taking the result up to house i-1 (dp[i-1]).
    - Robbing house i and adding its value to the maximum amount robbed up to house i-2 (nums[i] + dp[i-2]).

3. Base Cases:

- For the first house, you can only rob that house, so dp[0] = nums[0].
- For the second house, you can either rob the first house or the second house, whichever has more money, so dp[1] = max(nums[0], nums[1]).

Algorithm Outline:

1. Initialize Base Cases:

- If there are no houses (nums is empty), the maximum money is 0.
- If there is only one house, the maximum money you can rob is the value of that house.

2. Build the DP Table:

- Initialize a dp array to keep track of the maximum money that can be robbed up to each house.
- For each house from the third one onward, apply the recurrence relation to calculate the maximum money you can rob up to that house.

3. Result:

- The final result will be stored in the last element of the dp array, which contains the maximum amount of money that can be robbed from all the houses without alerting the police.

Detailed Example Walkthrough:

Example 1:

Input: nums = [2, 7, 9, 3, 1]

The problem will calculate the maximum amount of money that can be robbed without alerting the police by breaking down the problem step by step:

1. *House 1:* Rob house 1, money = 2.
   - dp[0] = 2
2. *House 2:* Rob either house 1 or house 2. The maximum is 7.
   - dp[1] = max(2, 7) = 7
3. *House 3:* Rob house 3 plus house 1, or skip house 3 and rob up to house 2.
   - dp[2] = max(7, 9 + 2) = 11
4. *House 4:* Rob house 4 plus house 2, or skip house 4 and rob up to house 3.
   - dp[3] = max(11, 3 + 7) = 11
5. *House 5:* Rob house 5 plus house 3, or skip house 5 and rob up to house 4.
   - dp[4] = max(11, 1 + 11) = 12

Output: The maximum amount of money that can be robbed is 12.

Time Complexity:

- **O(n):** We iterate through the list of houses only once, where n is the length of nums. For each house, we perform a constant amount of work, making the time complexity linear in terms of the number of houses.

Space Complexity:

- **O(n):** We use an additional array (dp) to store the maximum amount of money that can be robbed up to each house, which takes linear space. However, the space complexity can be reduced to O(1) by using only two variables to keep track of the last two values in the dp array, as we only need the results from the last two houses at each step.

Optimizations:

Space Optimization:

- Since we only need the last two values from the dp array at any point, the space complexity can be reduced from O(n) to O(1) by replacing the dp array with two variables to store the maximum amounts from the previous two houses.

Early Exit:

- If the input array nums has only one house, the problem can be solved in constant time O(1) by returning the value of that house directly.

Summary:

- This is a classic dynamic programming problem where each house gives two choices: rob it and skip the adjacent one, or skip it and move to the next house.
- The key is to maximize the amount of money while respecting the adjacency constraint.
- By storing the maximum money you can rob up to each house, you can calculate the final answer efficiently in linear time using a dynamic programming approach.